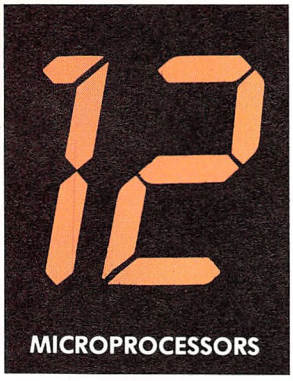


16-bit microprocessors

Why 16 bits?

Our discussions of 4 and 8-bit microprocessors have highlighted some of the advantages of using 16-bit devices: processing data in longer bit groups generally gives faster system operation, whether the system is used for control, communication or high speed computation; and a microprocessor with a longer bit length usually offers a more extensive instruction set with greater capability.

So, for any given operation, a 16-bit microprocessor can process twice to four times as much information as an 8 or 4-bit device, and the instructions are often more complete, resulting in shorter and more efficient programs.

Minicomputers, personal computers, 'smart' terminals, complex process controllers, display terminals, plotters, graphics terminals, high speed communications networks, word processing systems, special military systems, diagnostic systems and video games are all examples of 16-bit microprocessor applications.

Table 1
Possible design approaches to a 16-bit system

Approach	Advantages	Disadvantages
Microprocessor	Can be configured to meet any system requirements	Must be assembled into final system form
Microcomputer	Single chip systems with low assembly costs possible	Fixed input/output configurations. Memory is not easily expandable
Microcomputer modules	Pre-wired as a microcomputer module with provisions for memory and input/output expansion. Ideal for initial low cost start on small system	Relatively expensive if large numbers of identical systems are required

At the top of this list are the application areas that need high computational power, and this can be the overriding requirement for a 16-bit microprocessor. Most applications, in fact, need a fast, accurate numerical performance and are simply just easier to implement with 16-bit microprocessors (operating at impractically low speeds with 4 or 8-bit devices).

When a microprocessor or microcomputer element is selected for a given application, it is as well to bear in mind that although present requirements may not warrant a 16-bit device, future needs might. It is better, therefore, to design the system around a larger capacity element so that extra costs are not incurred in system redesign. Another advantage is that the same software can be kept as the system expands.

Advantages of different 16-bit devices

A 16-bit microprocessor can do all that a 4-bit or 8-bit microprocessor can do and more. When a 16-bit device is chosen, the main question to answer is what to buy: a microprocessor, a microcomputer or a microcomputer module? *Table 1* compares the advantages and disadvantages of these three different approaches.

Microcomputers and microcomputer modules offer certain advantages, however using a microprocessor means that the designer can structure the input and output to meet present and future system needs. The disadvantage, though, is that once such a design has been developed on paper, the circuit layout must be created, the parts assembled, and the results tested to achieve a working system. This can be a time consuming exercise, especially if only a few systems are to be built.

The main difference between a single package microcomputer IC and a microprocessor is that the memory and processor are *combined*, so there are fewer circuits to

assemble and wire together. However, because the memory and I/O are contained within the microcomputer, their structures are limited by the architecture of the device.

If the on-chip memory and I/O structure of the microcomputer are satisfactory for a given application, on the other hand, then this approach results in a simpler and less expensive system. The designer is still left with the problem of assembling and testing the final system, though.

The microcomputer module, on the other hand, is a completely wired and tested microcomputer containing a microprocessor, ROM (which sometimes contains a monitor program allowing user programs to be assembled and run), and a limited amount of RAM for data memory and program development (about 1 to 2K locations). There is, of course, also an I/O structure dedicated to reading in information, and sending it out to a display.

The advantage of a module like this is that it can be immediately used for program development and verification.

In many cases, the program needs only slight modification before immediate use in the intended systems application. This approach offers the quickest, simplest and sometimes cheapest method of building small numbers of systems.

However, there are disadvantages. Once the system is designed, the microcomputer module does not provide the cheapest way of building a large number of copies of a given system. Microcomputer modules are usually set up primarily as development tools. This may make it awkward to match the fixed module's structure to the final system's requirements.

However, the savings in time and effort needed to construct a working system, usually outweigh the limitations of structure and the initial cost disadvantages of the microcomputer module.

If many copies of a particular system are needed, then the primary design can be developed on the microcomputer module, and then converted to the appropriate microprocessor or microcomputer version. In this way a large number of relatively inexpensive systems can be developed. We shall follow this approach in this and the next chapter.

A typical 16-bit microcomputer module

The TM 990/U89 microcomputer board is fairly typical of this type of device, and an annotated photograph is shown in *figure 1*. The TM 990/U89's basic hardware features are summarised in the block diagram in *figure 2*.

Hardware features

The heart of the module is the TMS9980A microprocessor. Its behaviour is similar to that of the TMS9900 family of microprocessors and microcomputers, in that its instruction set operates on 16-bit words in memory.

In the case of the TMS9980A, data and instructions are transferred 8 bits at a time on the 8-bit data bus. This makes the basic architecture of the data bus similar to that of an 8-bit microprocessor, but as far as the system's programmer is concerned, it behaves as a *16-bit microprocessor*. Any program developed on the TMS9980A will happily run on the TMS9900 family of microprocessors and the TMS9970 microcomputer.

The TM 990/U89 microcomputer also has the following components:
1) 4K ROM/EPROM which contains a monitor – UNIBUG – which provides a symbolic assembler and the software interfacing to the input/output components on the board. There is room on the board to directly expand this ROM/EPROM memory by an additional 2K by simply mounting another memory circuit on the board.
2) 1K RAM which can be used for data or user program storage. There is room on the board for an additional 1K of RAM.
3) A keyboard that allows alphanumeric entry of program instructions, assembler commands and data into the RAM and input/output locations. This keyboard resembles that of an advanced scientific calculator, as many of the keys are devoted to alphanumeric characters and system commands. This means that full ASCII character entries are possible.
4) A 10-character LED display for displaying the information input through the keyboard or for displaying program and data memory information as commanded

by the keyboard entries. Not all alpha-numeric characters are displayed in true form, however, some are represented by symbols that *can* be displayed. Thus, it is called a **psuedo ASCII display**.

5) A piezoelectric speaker to output audio information, such as prompting tones to the programmer/user.

6) An audio cassette interface that allows the user to connect a cassette recorder to the module, in order to store or recall program or data information. This allows the user to add bulk memory easily and inexpensively.

7) A serial data interface for connecting the module to an external terminal for keyboard entry and printer output.

8) Memory and input/output expansion connectors to add additional memory and input/output boards to the microcomputer module, to expand memory and input/output to the full addressing capability of 16 Kbytes of the TMS9980A microprocessor.

While the TMS9980A's board structure and features tend to make it a system development and laboratory component, the expansion connector features enable the board to be used as a complete system's central microcomputer.

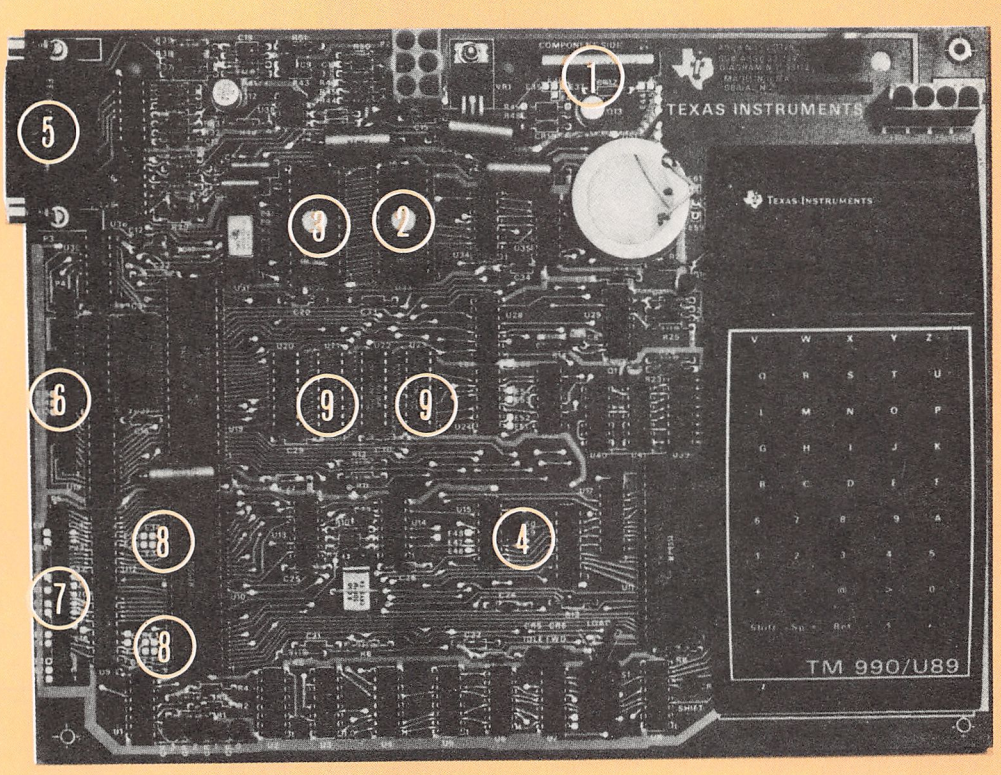
The designer simply has to define the on-board program memory, the additional memory and the I/O board designs to match the requirements of the system being built.

Once the program is developed, and the module is included in the system, the program can be written into the TM 990/U89, EPROM or ROM circuits. If there is insufficient memory space available, extra memory boards can be used.

1. The TM 990/U89 microcomputer board.

1

1. On-board cassette control
2. EPROM
3. EPROM expansion
4. Jumpers for 2K EPROM expansion
5. RS-232C port
6. Off-board data, address and control port
7. Parallel input/output port
8. Jumpers from LED modification
9. RAM expansion



Over page

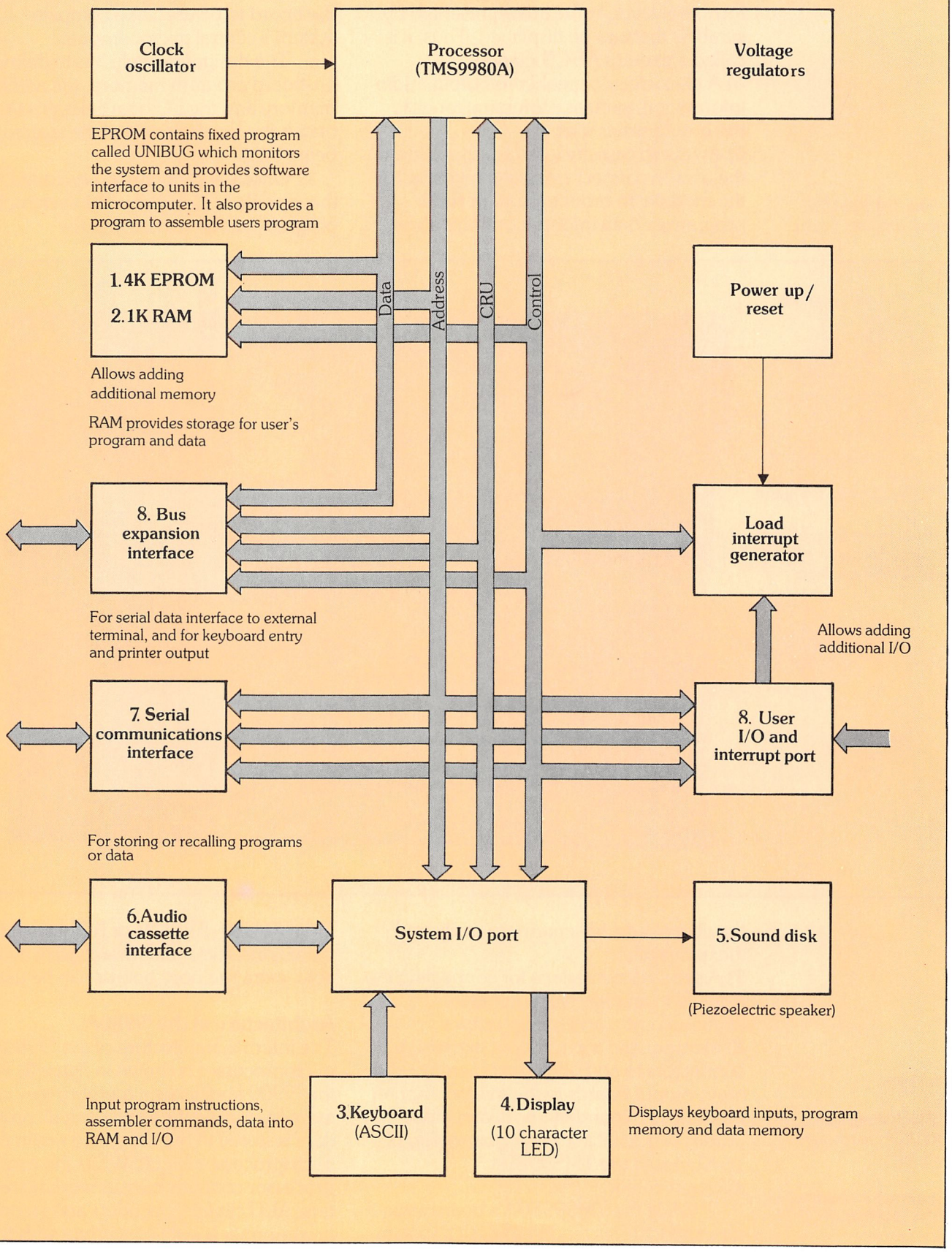
2. Block diagram of TM 990/U89's hardware features.

3. Internal architecture of the TMS9980A microprocessor.

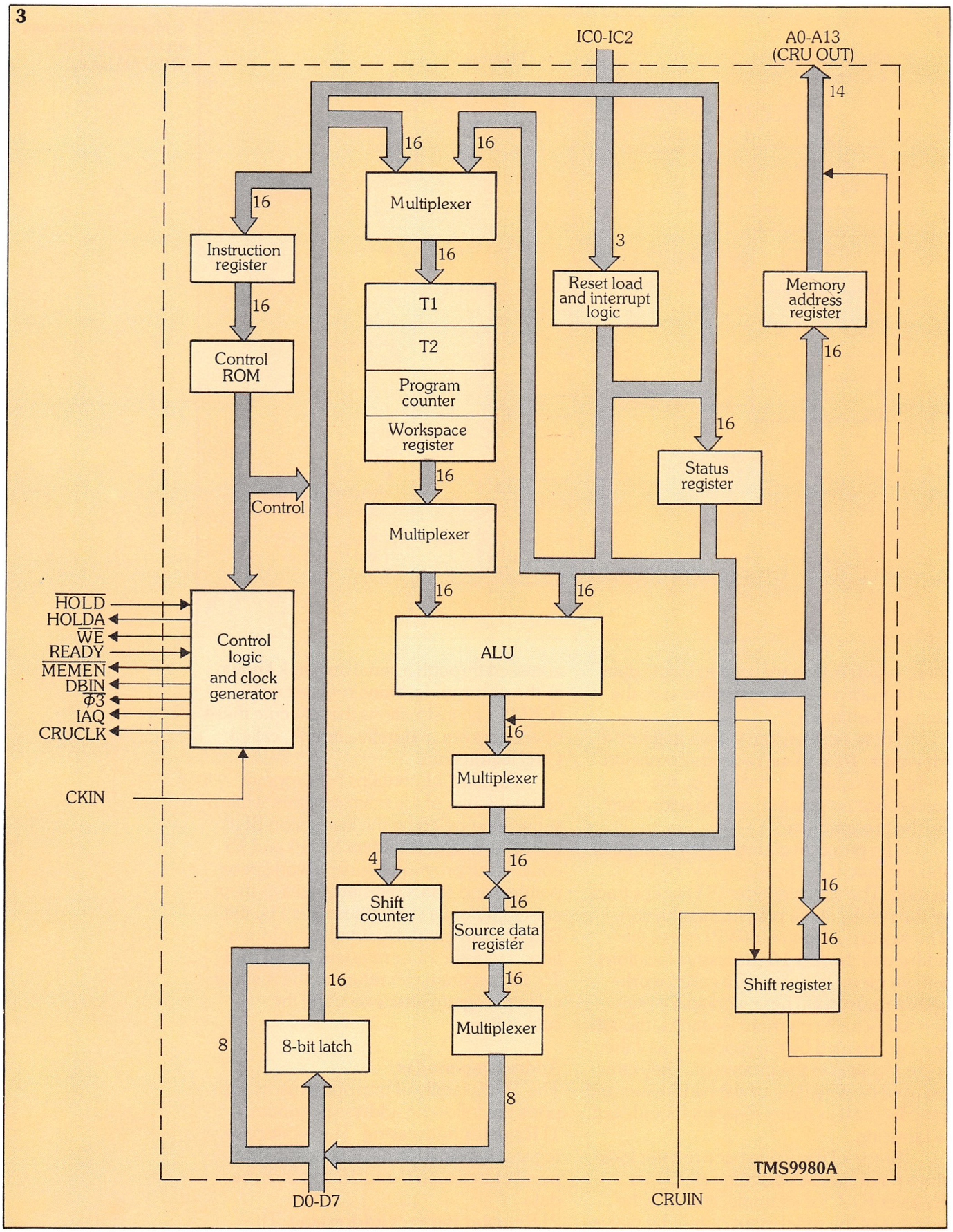
Architecture of the '9980A

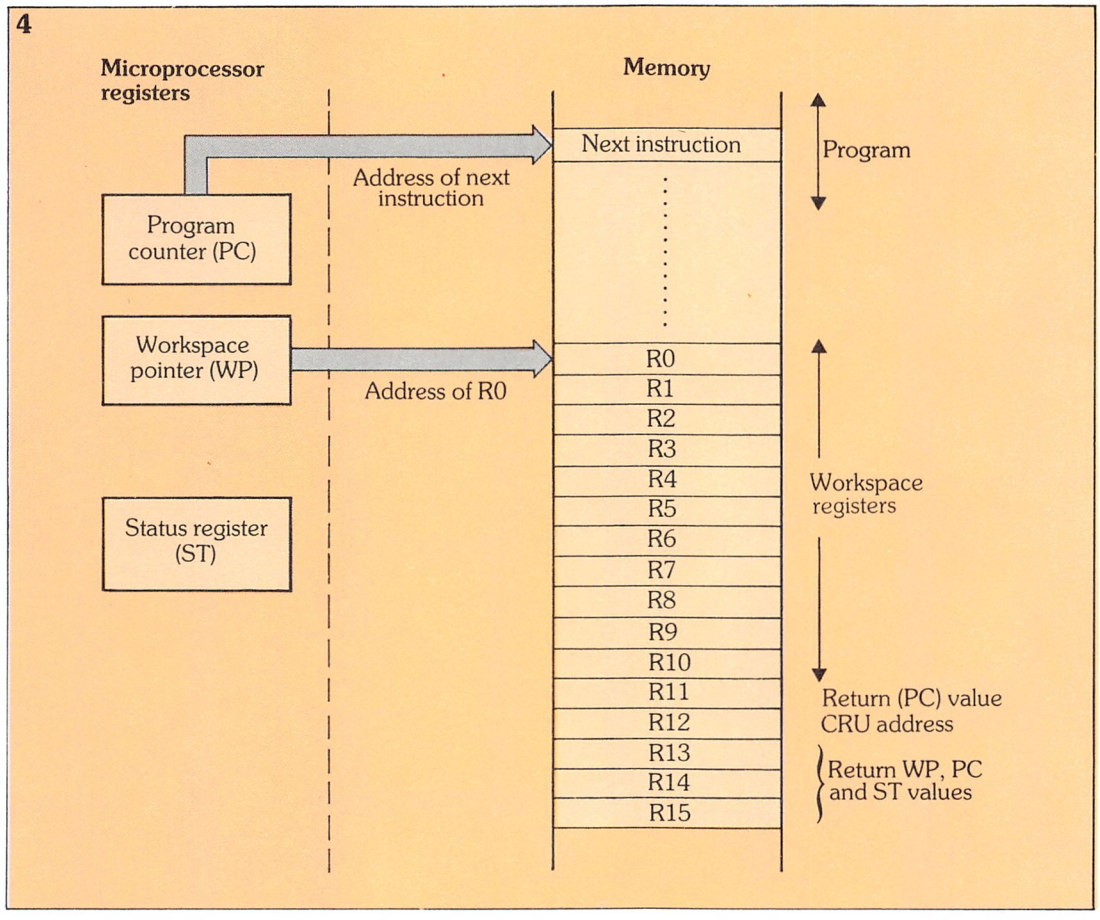
The internal architecture of the '9980A microprocessor is shown in *figure 3*. All of the '9900 family use a memory-to-memory architecture, in which all program data is stored in memory, and not in the microprocessor circuit. While the microprocessor contains various working registers, ALU circuits, decoders and so on, the programmer is only concerned with the

2


Clock oscillator
Processor (TMS9980A)
Voltage regulators
EPROM contains fixed program called UNIBUG which monitors the system and provides software interface to units in the microcomputer. It also provides a program to assemble users program
1.4K EPROM
2.1K RAM
Data
Address
CRU
Control
Power up/ reset
Allows adding additional memory
RAM provides storage for user's program and data
8. Bus expansion interface
Load interrupt generator
For serial data interface to external terminal, and for keyboard entry and printer output
Allows adding additional I/O
7. Serial communications interface
8. User I/O and interrupt port
For storing or recalling programs or data
6.Audio cassette interface
System I/O port
5.Sound disk
(Piezoelectric speaker)
Input program instructions, assembler commands, data into RAM and I/O
3.Keyboard (ASCII)
4. Display (10 character LED)
Displays keyboard inputs, program memory and data memory

3


IC0-IC2
A0-A13
(CRU OUT)
14
16
16
Multiplexer
16
16
Instruction register
16
Control ROM
3
Reset load and interrupt logic
Memory address register
16
T1
T2
Program counter
Workspace register
16
Status register
16
Multiplexer
Control
16
16
HOLD
HOLDA
WE
READY
MEMEN
DBIN
Φ3
IAQ
CRUCLK
Control logic and clock generator
ALU
16
Multiplexer
CKIN
4
16
16
Shift counter
16
Source data register
16
16
16
Shift register
16
Multiplexer
8
8-bit latch
8
TMS9980A
D0-D7
CRUIN



4. Memory-to-memory architecture of the TMS9900 family.

three registers that are used to locate data and instructions in memory and save the status information.

These registers are shown in *figure 4*. However, 16 working registers contained in the workspace are defined by the workspace pointer and can be addressed by the programmer.

The program counter (PC) maintains the address of the instruction to be executed. The status register (ST) keeps track of the results of the previous instruction. The workspace pointer (WP) stores the address of the first 16 successive locations in data memory, which are called **workspace registers**. These workspace registers are actually memory locations, though they are treated by the processor as if they were registers in the processor. They can be used to store data or data addresses and may be used as index registers for indexed addressing.

Some of the workspace register locations have special dedicated purposes. Register R12, for example, contains the address of the serial input/output bit that is accessed through a serial data link known as the **communications register unit (CRU)**. The instructions that execute these operations are, naturally enough, called CRU instructions.

Register 11 contains the program counter value of the main program while a subroutine called by the instruction BL is being executed. Registers 13, 14 and 15 contain the program counter, workspace pointer, and status register values as they existed before a subroutine called by the BLWP instruction. By using the information in register 11 or registers 13, 14 and 15, the program can return to the main or calling program after executing the subroutine.

Addressing modes

The '9900 family of microprocessors supports the following addressing modes:

1) **Register addressing**. The workspace register indicated in the instruction by a number between 0 and 15 contains the data.

2) **Register indirect addressing**. The

workspace register indicated by an asterisk (*) followed by a number between 0 and 15 is used as an address register to hold the address of the data. If the number is followed by a plus (+) in the instruction mnemonic coding, the contents of the address register are incremented following the instruction. This causes the address register to act as a data counter that contains the address of (i.e. it points to) successive data locations as the instruction is executed.

3) **Direct addressing**. The address in the instruction is the location in memory to be used by the instruction. The mnemonic coding used to indicate this type of addressing is @, followed by the direct address value of the symbol.

5. Examples of the addressing modes used by the TMS9900 family of microprocessors.

5

Addressing mode	Mnemonic coding for CLR instruction	Effect of instruction if R1 contains 100_{16}
Register	CLR R1	R1 cleared to all zeros
Register indirect	CLR *1	Location 100_{16} is cleared to all zeros, R1 not affected
	CLR *1+	Location 100_{16} is cleared to all zeros. The contents of R1 are incremented to 102_{16}
Index	CLR@>10(R1)	The location $100_{16} + 10_{16}$ or 100_{16} is cleared to all zeros. R1 is not changed
Direct	CLR@>200	The location 200_{16} is cleared to all zeros

Notes: the symbol > denotes a Base 16 (hexadecimal) value

4) **Indexed addressing**. The data address is formed by adding the offset contained in the instruction to the value contained in the index register specified (registers 1 to 15). The mnemonic coding for this type of addressing is @, followed by the offset value or label, followed by the register number in parenthesis.

Some examples of the mnemonics used to code the addressing modes of the CLR (clear) instruction in assembly language, are shown in *figure 5*. This instruction causes 16 zeros to be sent to the location specified by the addressing mode in the instruction.

Instruction set

The instruction set of the TMS9900 family is summarised in *table 2*. Most of the operations use all of the addressing modes listed, although there are exceptions. For example, immediate operations (those with mnemonics ending in I) use immediate addressing to indicate the data constants to be loaded into registers. Data is contained in program memory as the second word of the instruction.

Register addressing is used to locate data variables from the workspace register involved in an instruction. Both multiply and divide instructions use register addressing for the destination data locations, that is, the product or quotient/remainder are stored in two successive workspace registers. Shift operations only use register addressing.

A special kind of indexed addressing, **program-counter relative addressing**, is used by the jump instructions: to determine the instruction address to which the program jumps, the displacement contained in the instruction is added to the contents of the program counter (acting as the index register).

Arithmetic instructions

The '9900 family's arithmetic operations include addition and subtraction (byte, 8-bit, or word, 16-bit, operations), negation, absolute value, increment and decrement (by one or two), and multiplication and division. (The absolute value of a number, remember, is the unsigned value of that number.)

The data movement operations include byte and word movement instructions, and instructions to store the status and workspace registers.

Initialisation operations include clearing a data word to all zeros, and setting a data word to all ones. The workspace pointer, interrupt mask (this is a special interrupt control) and workspace registers can all be loaded with constants from the program with load instructions.

Logical operations include the OR, AND, invert and Exclusive-OR operations, as well as some special logical OR operations (SOC, SOCB).

Memory data can be compared to a

Table 2
TMS9900 family's basic instruction set

Arithmetic operations	
Addition	A, AB, AI
Subtraction	S, SB
Negation	NEG
Absolute value	ABS
Multiply (16 bit × 16 bit)	MPY
Division (32 bit + 16 bit)	DIV
Increment	INC, INCT
Decrement	DEC, DECT
Logic operations	
Invert	INV
OR	ORI, SOC, SOCB
AND	ANDI, SZC, SZCB
Exclusive-OR	XOR
CRU I/O	
Single bit	TB, SBO, SBZ
Multiple bit	STCR, LDCR
Data movement	
Move data	MOV, MOVB, SWPB
Store registers	STST, STWP
Compare operations	
Compare to constant	CI
Compare	C, CB
Masked compare	COC, CZC
Initialisation	
Clear	CLR
Set (all ones)	SETO
Load immediate	LI
Load registers	LWPI, LIMI
Special operations	
Execute instruction out of sequence	X
Extended operation	XOP

Branch operations	
Unconditional branches	B, JMP, RTWP
Subroutine call	BL
Context switch	BLWP
Conditional branches	JEQ, JNE, JGT, JOP, JOC, JNC, JNO, JLT, JH, JHE, JL, JLE
Addressing Modes	
Register	
Register indirect, without auto-incrementing	
Register indirect, with auto-incrementing	
Indexed	
Direct	
Immediate	
Relative (used in jump instructions)	
Control	
Reset	RSET
Idle condition	IDLE
Shift Operations	
Arithmetic shifts	SLA, SRA
Logical shift right	SRL
Circular right shift	SRC

Meaning of mnemonic endings: B, Byte; I, Immediate; T, by two.

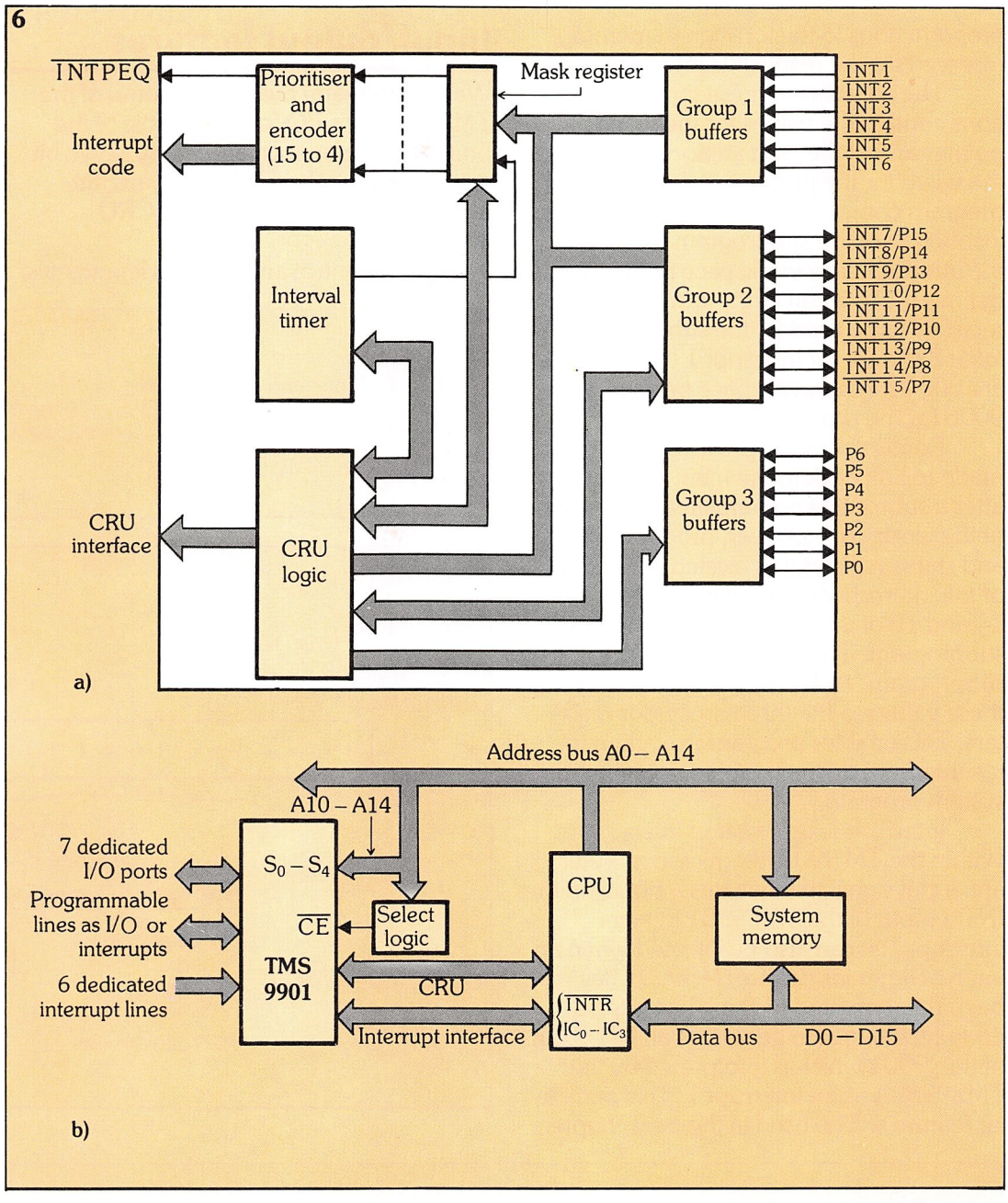
constant or to another memory byte or word. Again, special compare provisions are provided in COC and CZC to select specific word or bits in order to check system limits or the status of subsystems.

The shift operations include arithmetic shifts (left or right), logical shift right and a circular right shift (right circulate).

There are two types of subroutine calls and two unconditional branches. A total of twelve conditional jump operations check status bits for conditions of: carry; parity; equal; arithmetic and logical greater than and less than; and so on.

Special instructions exist which move data serially over a serial input/output data link – the **communications register unit** (CRU). The CRU and its instructions enable the same kind of serial data input as SAM (simplified architecture microprocessor). Individual bits can be tested at an input with TB, or set or cleared with SBO or SBZ. Multiple bits (and the number can be programmed) can be stored in memory from the CRU or loaded into the CRU with the STCR or LCDR instructions, respectively.

All of these instructions are available



6. (a) Internal architecture of the TMS9901 programmable systems interface IC; **(b)** how the TMS9901 is connected to the microprocessor.

on the TM 990/U89 microcomputer module: programmers entering the instructions directly, in mnemonic form.

Interrupt features

Unlike the TMS8080A which only has one level of interrupt, the TMS9980A has eight such levels. Each level is assigned a priority, and any level can be **masked off** (made inactive) by the programmer.

A 3-bit interrupt code, IC_0, IC_1 and IC_2, indicates the interrupt levels to the microprocessor. These signals are generated by a TMS9901 programmable systems interface IC which is fixed in place on the TM 990/U89 board. The internal architecture of the TMS9901 is illustrated in *figure 6a*; *figure 6b* shows how it is connected to the microprocessor. Using this device, the programmer can define which interrupts are to be active, and what they should mean.

The TMS9980A responds to a given interrupt by performing what is known as a **context switch**. This is a form of subroutine jump to a subprogram designed to service that interrupt. When the interrupt occurs and the context switch is made, the

program goes to the location in memory reserved for the respective interrupt code.

The contents of these memory locations contain the values for the workspace pointer and at the next adjacent location the value for the program counter. The program counter value addresses the first instruction of the subprogram to service the interrupt. The workspace pointer value defines the location of the workspace registers to be used by the subprogram. As *table 3* shows, for interrupt 1 the locations are 0004 for the workspace pointer, and 0006 for the program counter.

Remember, that provision must be made to link back to the main program after a subroutine is completed. The values of the workspace pointer, program counter and status register that existed at the time of the interrupt are saved in registers 13, 14 and 15 of the new workspace for the subprogram. Then, at the end of the subprogram, an RTWP instruction restores these values to the three processor registers. This enables program execution to resume from the interrupted point in the system program.

Interrupt 1, is of special interest. Within the TM 990/U89 microcomputer, any input signal that initiates a signal on the INT 3 input of the TMS9901 triggers interrupt 1; it can also be initiated when an interval timer inside the TMS9901 has been decremented to zero. The interval timer reaching zero generates an INT 3 out of the '9901 which is interpreted by the TMS9980A as an interrupt 1. This particular feature will be used in the next chapter.

Input/output features

We have already mentioned some of the TM 990/U89's I/O features. Most of the board's components, and many single bit control or input lines off the board, are accessed by the TMS9980A's CRU instructions.

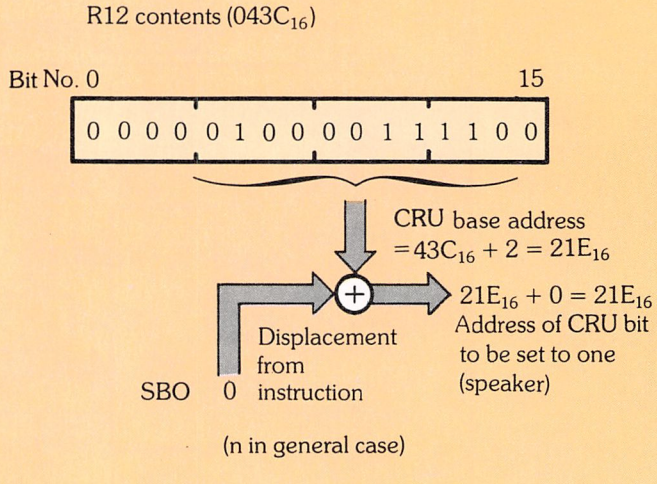
Both control and address line signals

7. Instruction sequence to turn on the piezoelectric speaker on the TM 990/U89 board.

7

Label	Mnemonic	Operand	Comment
	LI	12,>43C	Set up $43C_{16}$ (twice $21E_{16}$) in R12
	SBO	0	Set bit $21E_{16} + 0$ to a one

8



8. Overall addressing procedure for the piezoelectric speaker.

Table 3
Interrupt codes and their memory locations for the TMS9980A

Interrupt code IC_0 IC_1 IC_2	Function	Locations of address of 1st instruction (PC value)	Location of workspace pointer (WP value)
0 0 0	Reset	0002	0000
0 0 1	Reset	0002	0000
0 1 0	Load	3FFE	3FFC
0 1 1	Interrupt 1	0006	0004
1 0 0	Interrupt 2	000A	0008
1 0 1	Interrupt 3	000E	000C
1 1 0	Interrupt 4	0012	0010
1 1 1	–	–	–

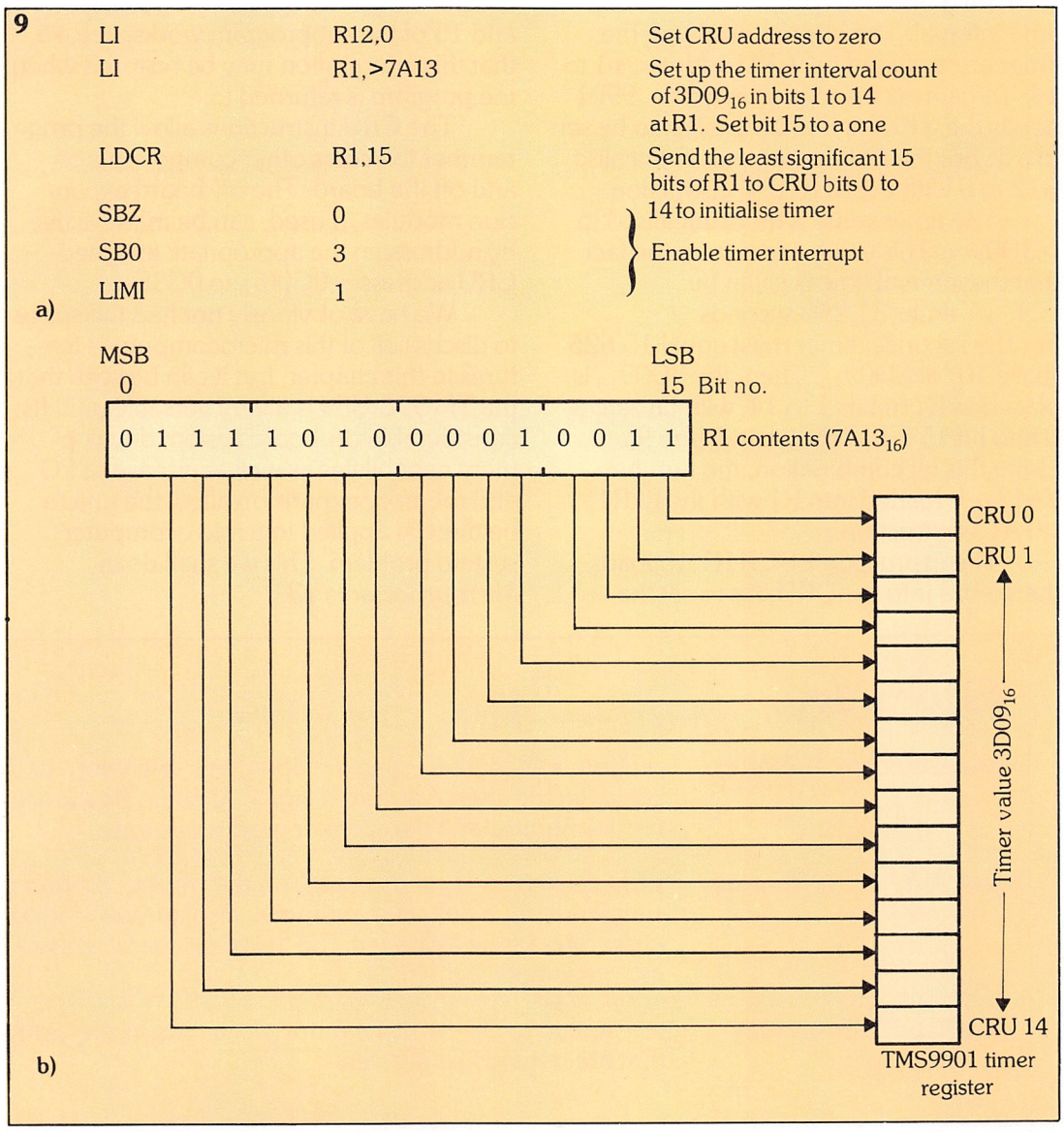
are used to locate I/O units as well as memory cells. Each individual bit in the system input/output structure is assigned a CRU address, in the same way as memory locations are assigned addresses. Each CRU address is related to a base address of a particular I/O unit. These bits can be accessed one bit per input/output instruction, using the single bit CRU instructions SBO (Set CRU Bit to One), SBZ (Set CRU Bit to Zero) and TB (Test value of CRU Bit). The address of the accessed bit is kept in a special position – bits 4 to 14 of workspace register 12.

A single bit example

As an example of such an operation, the piezoelectric speaker (mentioned earlier in the chapter) is assigned the CRU address $21E_{16}$ in the TM 990/U89 system. So, to turn the speaker on, the instruction sequence in *figure 7* is used.

The signed displacement in the instruction (0 in this case, n in the general case) is added to the CRU address in R12 (half of the value loaded into R12) to form the address of the CRU bit to be set to 1. The CRU address is half the value in R12 because bit 15 is disregarded. (This is just like shifting bits right one position which, remember, divides a binary number by 2.) The overall addressing procedure is shown in *figure 8*.

If the instruction had been SBO 5, then 5 would have been added to the base address in *figure 8* to set the 6th bit to a 1.



9. (a) Instruction sequence to load TMS9901 timer; **(b)** the effect of instruction LDCR R1,15 (setting CRU bit 0 provides access to timer register).

Similarly, an SBZ 0 instruction with $43C_{16}$ in R12 turns the speaker off.

Multiple bit example

The individual CRU bits can be accessed several bits at a time with the multiple bit CRU instructions: LDCR (send the number of bits specified to successive CRU locations, starting at the base address contained in R12); and STCR (receive the number of bits specified from the CRU, starting at the location specified in R12, and send the successive bits to the memory location specified in the instruction).

As an example of such an operation, *figure 9a* illustrates the procedure for initialising the '9901's interval timer in the TM 990/U89 microcomputer at position U10, for a 500 millisecond (0.5 second) time interval. The bits for initialising the timer are assigned the CRU addresses 1 to 14. To gain access to the timer, the '9901 needs the CRU bit 0 to be the first to be set to a 1. So, the first operation is to initialise R12 to 0 with the LI R12, 0 instruction.

The timer value is them initialised to $>3D09_{16}$. This value arises from the fact that the interval time is given by:

T = timer/31,250 seconds

For 0.5 seconds, timer must equal 15,625 (base 10) or $3D09_{16}$. Then, the $3D09_{16}$ is placed in R1 in bits 1 to 14, with bit zero a 0 and bit 15 a 1 as shown in *figure 9a*. Using this bit combination, the number $7A13_{16}$ is loaded into R1 with the LI R1, >7A13 instruction.

The instruction LDCR R1,15 loads the 15 bits into the CRU bits to set the timer register in the TMS9901. The 1 in CRU bit zero gains access to the timer.

The three remaining instructions shown in *figure 9a* are required to enable the INT 3 signal (generated by the internal timer in the TMS9901 going to zero) to pass through to the TMS9980A as an interrupt.

When the timer decrements to zero 0.5 seconds later, the interrupt level 1, properly enabled, causes the program to jump to the subprogram address contained in memory location 0006. At the same time, it sets up the 16 workspace registers defined by the workspace pointer value contained in memory location 0004.

The contents of the current workspace pointer, program counter and status register values are saved in registers 13, 14 and 15 of the subprogram workspace, so that this information may be restored when the program is returned to.

The CRU instructions allow the programmer to access other components on and off the board. The off-board expansion modules, if used, can be made active by addressing the appropriate assigned CRU addresses $0C00_{16}$ to $0C3E_{16}$.

We have obviously not had the space to discuss *all* of this microcomputer's features in this chapter, but it can be seen that the TM 990/U89 is a very versatile unit. Its considerable on-board system development capabilities as well as its ease of I/O and memory expansion allow the unit to be directly applied to a microcomputer system problem. This we shall do in *Microprocessors 13*.

Glossary

piezoelectric speaker	output transducer, whose operating element is made from a piezoelectric crystal. When a signal of some frequency is applied, the crystal vibrates and the speaker produces a tone
pseudo ASCII display	LED display used, in this case, on the microcomputer module. This is simpler than a full alphanumeric display and cannot show all the characters of the alphabet. The 'missing' characters are represented by other symbols
workspace registers	not strictly registers, but memory locations in the workspace memory that are treated like registers

Data networks

A history

Data networks have existed for only about fifteen or twenty years. Their purpose is to connect computers and other digital devices so that information can be transferred from one location to another. They act 'in the background', and users are generally unaware of their existance.

One of the first data networks was developed in the U.S. to link computers at the Lincoln Laboratories and Systems Development Corporation: users at each location were able to access facilities at the other site. The advantages of this link were quick to be noticed:
1) Certain facilities at one site need not be available at the other site as long as the network could be easily accessed.
2) Expensive facilities need only be available at one site – but both sites could share the cost.

Despite these obvious advantages, the world's first viable data network linking multiple locations was not built until 1970. The reason for this delay was that most computing systems in the 1960s were large mainframe systems. Generally, only large organisations could afford such machines and they tended to concentrate all their computing effort in centralised data processing sites which did not require any outside facilities.

Historically, the network built in 1970 is very important – it was the first to introduce the concept of packet switching. **ARPANET** (built by the Advanced Research Projects Agency of the U.S. Department of Defense) initially linked only four sites, but by 1971 it covered 15 sites in the U.S. and incorporated 25 computers (*figure 1*). By the late 1970s it linked 64 computers – the maxium it had been designed to take.

Access **nodes** into ARPANET were known as **interface message processors** (IMPS) – these were computers which interfaced users' computers to the network. The Honeywell-316 was the first type of computer used as an IMP.

During ARPANET's ten year operational life, computers changed dramatically and minicomputers and microprocessor based computers were introduced. Although cheaper than their mainframe equivalents they did not yet possess the level of processing power required. However, users began to recognise that by linking a large number of small computers with say, one or two larger mainframes, a very economical method of computing could be created; the smaller computers could take advantage of the processing facilities of the mainframe computer without the user needing to be in the central computer department of the building. Indeed, there was no need for the user to even be in the same building.

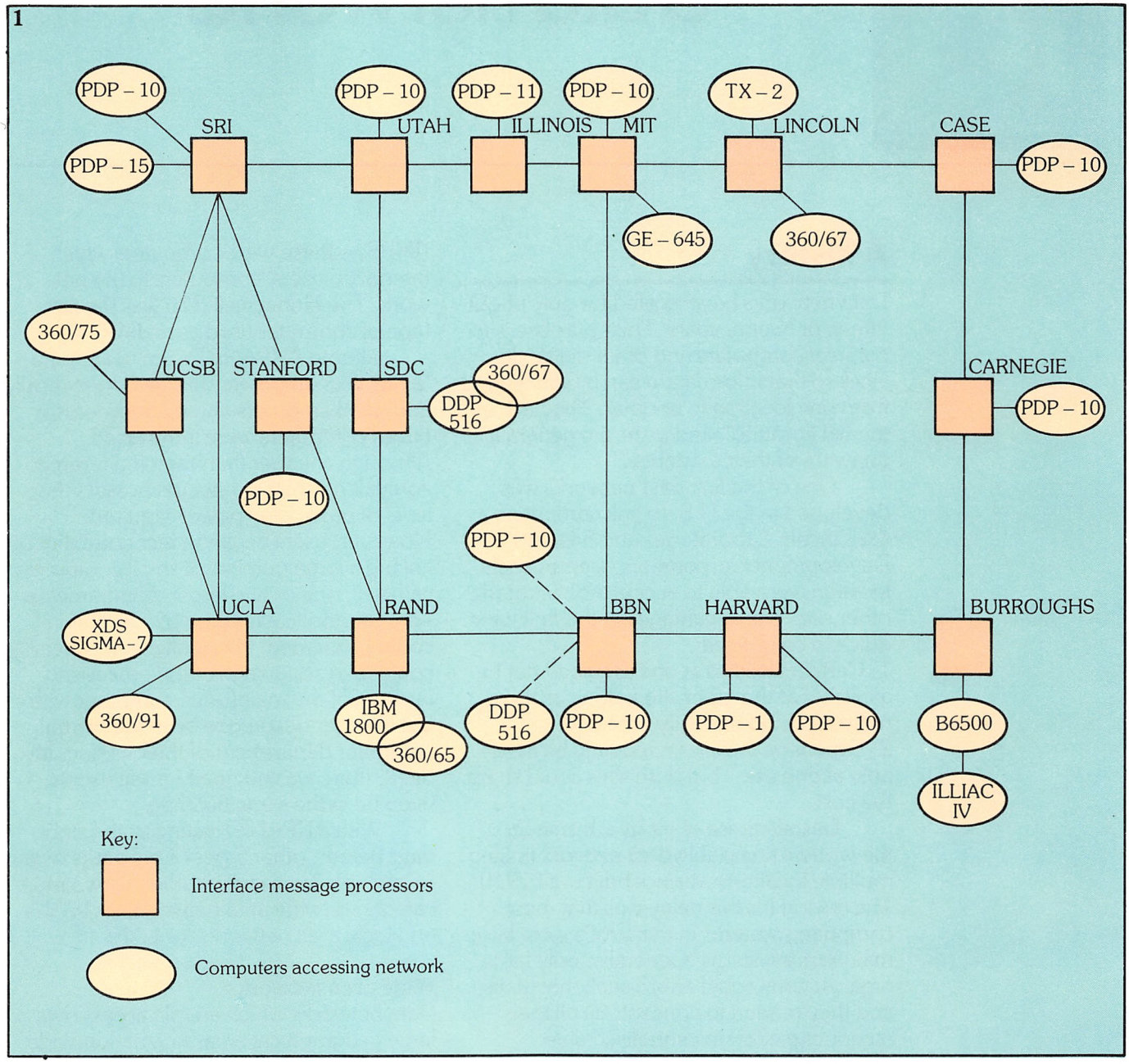
With ARPANET acting as the standard bearer, other types of networks began to emerge. These fall into one of two main categories: wide area networks, or WANs, and local area networks, or LANs.

Wide area networks

Data networks which enable access over a large geographical area, say, nation-wide or world-wide, are known as **wide area networks** (WANs). ARPANET was the first example of a wide area network, although distinctly limited in its use by the maximum number of users, 64.

The second major wide area network, known as **TELENET** and built in the U.S. in 1975, was far more versatile. TELENET was the first wide area network designed for public data communications. Like ARPANET, it was based on a packet switching model, but had the advantage of an *unlimited* number of users.

In the U.S. communications are regulated by the Federal Communications



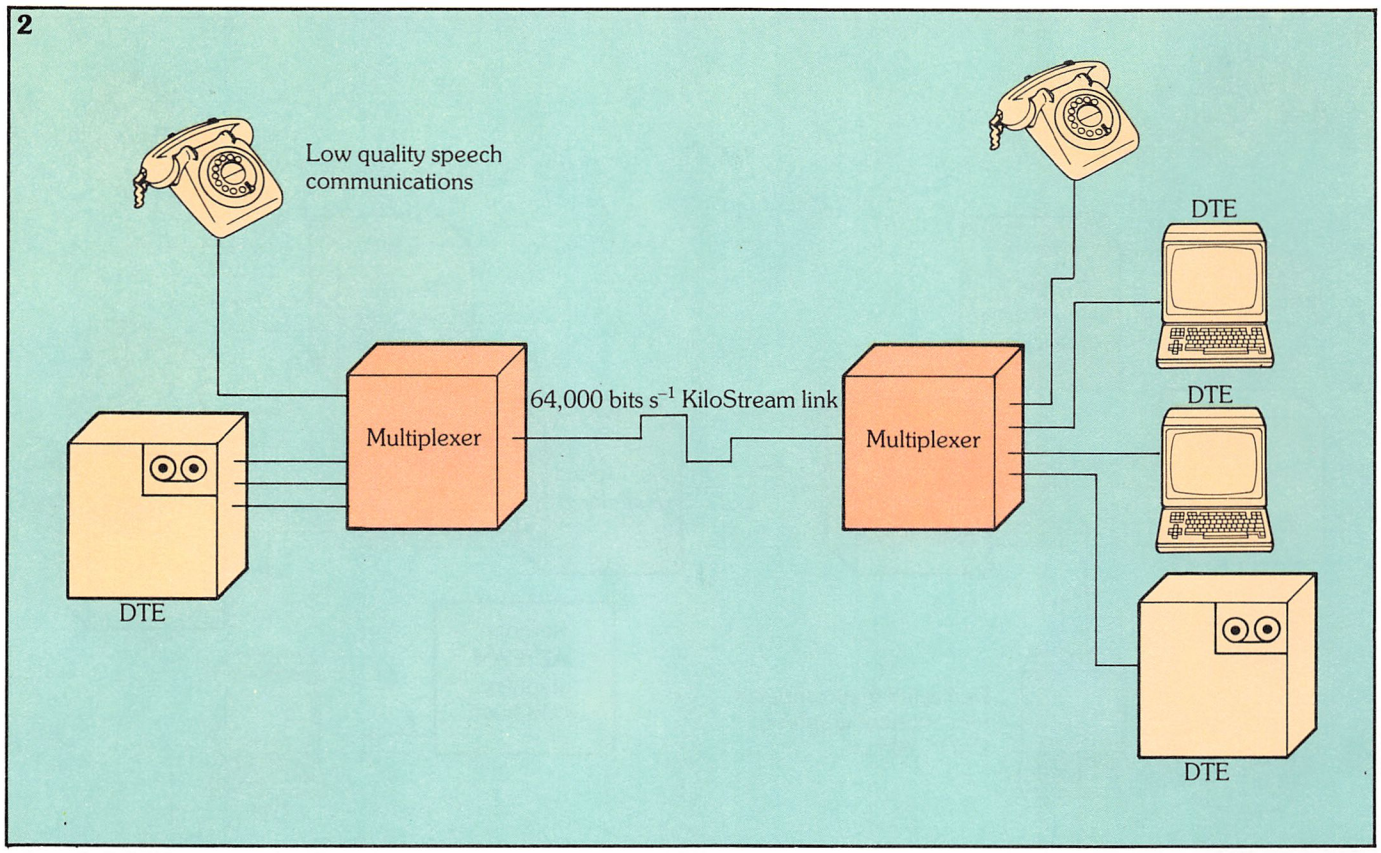
Commission. Groups wishing to provide a communications service file a request with the Commission which, if granted, allows them to implement the service. Such groups are known as **common carriers** – they are not able to monopolise a particular type of service as the Commission controls their functions.

In Europe (and many other countries), on the other hand, government appointed bodies known as the **PTT**s (for postal, telegraph and telecommunications authorities) have, until recently, controlled the communications networks. In the U.K., for example, the Post Office was the PTT until British Telecom was formed as a separate body. BT still, however, has a monopoly regarding telecommunications networks and services.

Recent legislation is stripping BT of its monopoly and private, competing groups are now being allowed to tender for telecommunications services. This is also happening in other countries, bringing control procedures more into line with those in the U.S.

Monopolies have a tendency to introduce new services only slowly. For example

1. In 1971, ARPANET covered 15 sites and linked 25 computers.



2. The KiloStream digital, point-to-point communications link which provides data signalling rates up to 64,000 bits s^{-1}.

although TELENET was in operation in the U.S. in 1975, British Telecom's packet swtiched data network, Packet SwitchStream, was not operational until 1981. Prior to this, data networking could only be undertaken on a wide area basis, via the PSTN – using modems. The existing PSTN is a network designed specifically for use with speech and, as we have seen in earlier *Communications* chapters, cannot provide the high signalling rate data communications suitable for data networks. The argument for de-monopolisation is that new services will be developed and in use far more rapidly.

Local area networks

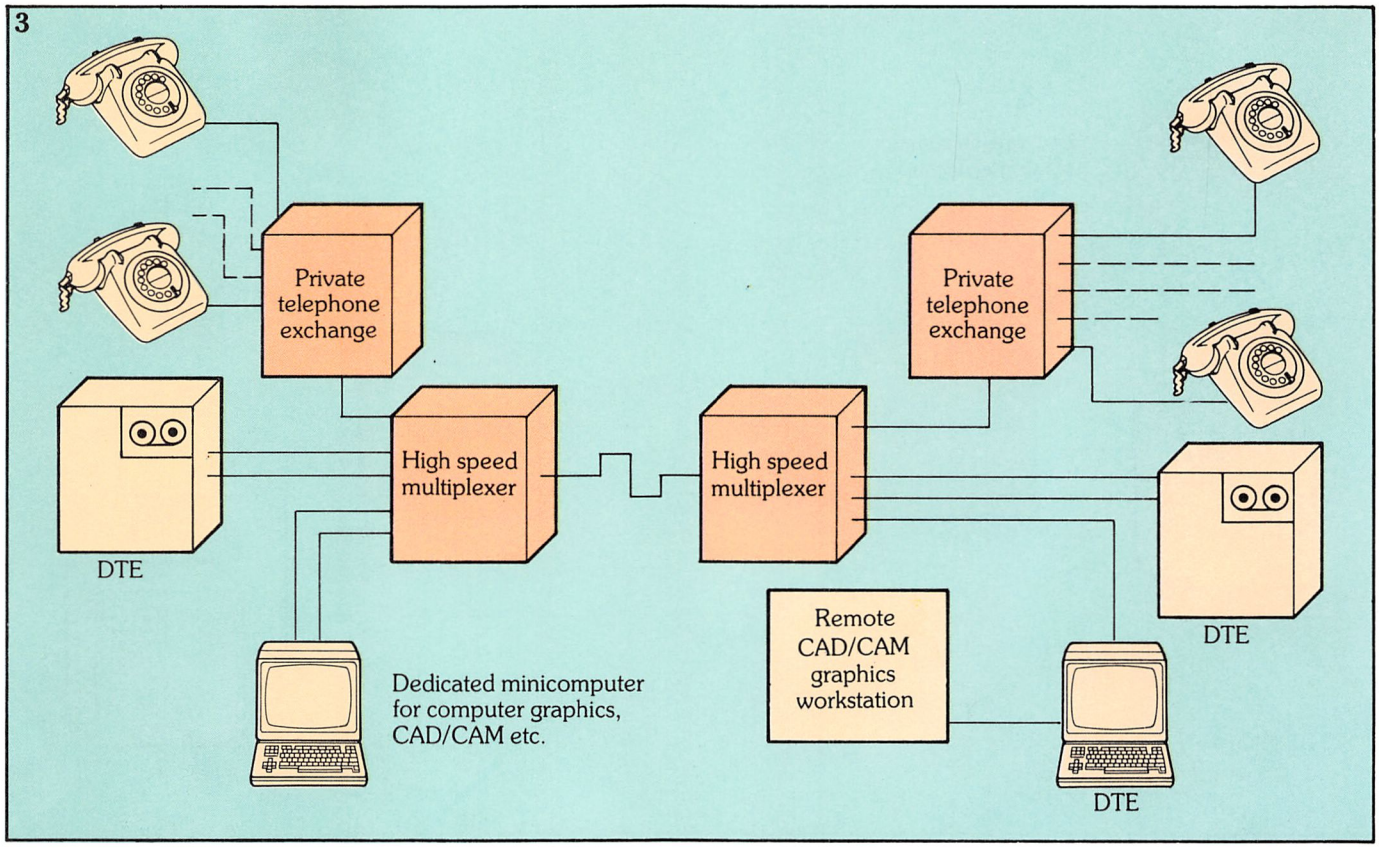
Wide area networks provide data networking services over a large geographical area – at the other end of the scale, networking over a small area is the province of **local area networks** (LANs).

Generally, local area networks are not provided by the PTTs, but are installed by private organisations, often as a means of linking their own data equipment. In the absence of any overall governing body ensuring that a single standard is maintained, this has led to the emergence of many machine specific LANs. Fortunately, however, recent attention has focused on two general types of local area networks: a broadcast network, e.g. Ethernet, and a ring structured LAN, e.g. Cambridge Ring. We shall be looking at these in greater detail in a later chapter.

Other types of wide area network

We have seen how the PSTN may be used as a wide area circuit switched data network, albeit allowing relatively poor data signalling rates and reliability. We have also seen how Packet SwitchStream provides a faster, more reliable national data network, based on packet switching principles.

These two examples illustrate quite well how it is difficult to provide a single telecommunications service which effectively combines voice and data. On the one hand, a circuit switched telephone network, designed specifically for speech communications, may be modified to transmit data, but a totally effective and reliable data network is not possible using this method. A packet switched data net-



work, on the other hand, is not at all suitable for speech communications, although ideal for data communications.

Recently, some new digital transmission communications services for speech *and* data have been introduced. In the U.K., these services fall into two categories, labelled by British Telecom as **KiloStream** and **MegaStream**.

KiloStream is a point-to-point service, i.e. much like a dedicated, leased line between sites, but is fully digital. The service is available at a number of data signalling rates: 2400, 4800, 9600, 48,000 and 64,000 bits s^{-1}. *Figure 2* illustrates a possible two site data communications system, joined by a 64,000 bits s^{-1} KiloStream link. Low quality speech transmission is possible.

At present, KiloStream has two major disadvantages:
1) it *is* point-to-point;
2) few access points are available.
However, switched networking solutions have been proposed and more and more exchanges are being equipped with the KiloStream facility, so that the disadvantages will be minimised in the future.

MegaStream

Like KiloStream, MegaStream is a fully digital communications service, although operating at a much higher basic data signalling rate: 2.048 Mbits s^{-1}; rates of up to 140 Mbits s^{-1} are planned. MegaStream also has the capability to be enhanced into a switchable network although this is not possible yet.

The high data signalling rate provided by the MegaStream service is useful in its ability to allow transmission of digitised speech, data or, indeed, any kind of digitised information, e.g. facsimile, teletex, television, video conferencing. *Figure 3* illustrates a possible MegaStream link between two sites and shows how many different services may be combined.

The three main digital data services we have looked at, Packet SwitchStream, KiloStream and MegaStream, form the core of a group of services offered by British Telecom, known as **X-Stream**.

Other services included in X-Stream are a small dish satellite communications system known as **SatStream** (which provides communications links to remote sites, of 64,000 bits s^{-1}) and an **integrated**

3. MegaStream operates at a very high data signalling rate – 2.048 Mbits s^{-1} – and allows the transmission of *all* types of digital information.

digital access service (IDA – which is really the first stage in the digitisation of the PSTN).

As an eventual aim, it is intended that all three wide area communications networks will be integrated together into a single digital service, known as the **integrated services digital network** (ISDN).

The integrated services digital network will support the digitised speech requirements of a circuit switched network, while also allowing packet switched data communications. Transmission of all other information such as video, facsimile etc. will be possible using one of these two switching methods on the ISDN.

Glossary

ARPANET	one of the first wide area data networks, based on the principles of packet switching
common carriers	private U.S. companies providing communications services. Their overall activities are governed by the Federal Communications Commission
integrated digital access (IDA)	first stage service in the replacement of the existing analogue based PSTN with a fully digitised system
integrated services digital netowrk (ISDN)	the future PSTN, based on purely digital transmission techniques, allowing integrated access by a number of communications services
KiloStream	digital communications service providing data signalling rates up to 64,000 bits s^{-1}
MegaStream	digital communications service presently providing a data signalling rate of 2.048 Mbits s^{-1}. Rates of 140 Mbits s^{-1} are planned
PTT	postal, telegraph and telecommunications authorities. Government appointed bodies, common to European countries, supervising communications networks. Recent legislation in many countries is eroding their monopoly over communications services

Using the TM 990/U89

Program development

As the TM 990/U89 microcomputer module is a self contained computer, it can be applied to any system problem that is compatible with its operating speed and storage capabilities. In fact, a number of different tasks can be handled by using the interrupt signals to notify the microcomputer that a job needs attention.

As a simple example, we'll consider using the TM 990/U89 as a clock, that chimes on the hour. Of course, the system could be made more complex, say, by designing it to play certain tunes on the quarter hours, maintain an hours and minutes display and chime and play a tune on the half hour. The work behind the creation of these more complex features can be gauged by an appreciation of what's involved in making the clock simply strike on the hour.

Since a microcomputer module is being used, the program is the only thing that we have to consider. Remember, the module already carries a piezoelectric speaker and interval timers on its board.

System flowchart

Figure 1 shows a possible flowchart of the system's activities. First of all the clock has to be set to the right time. Then the programmer must start the execution of the program by setting the system's TMS9901 interval timer to 0.5 seconds, and waiting for the timer to interrupt the microprocessor with a level 1 interrupt. The microprocessor can, of course, be handling other system operation tasks in between interrupts.

When the interrupt does occur, the 0.5 second counter is decremented. If it reaches zero the clock must chime; if it is not zero, then the 0.5 second interval timer in the TMS9901 is re-initialised. By initially setting the 0.5 second counter to an appropriate value, the clock is therefore made to chime every hour when the counter reaches zero.

The chiming is carried out by sending different tones through the microcomputer module's speaker. If we assume the chime tones to be 1 kHz notes, then we need to set the speaker to be switched on for 0.5 ms and then off for 0.5 ms. If this action is repeated, the speaker will be turned on and off 1000 times a second – giving us a 1 kHz note. This note will sound for one second, followed by one second's silence, once for each hour passed, on the hour.

Once all the required tones have sounded, the 0.5 second counter is re-initialised and the sequence repeated. The tone sounding sequence uses software controlled microprocessor timing loops.

Data requirements

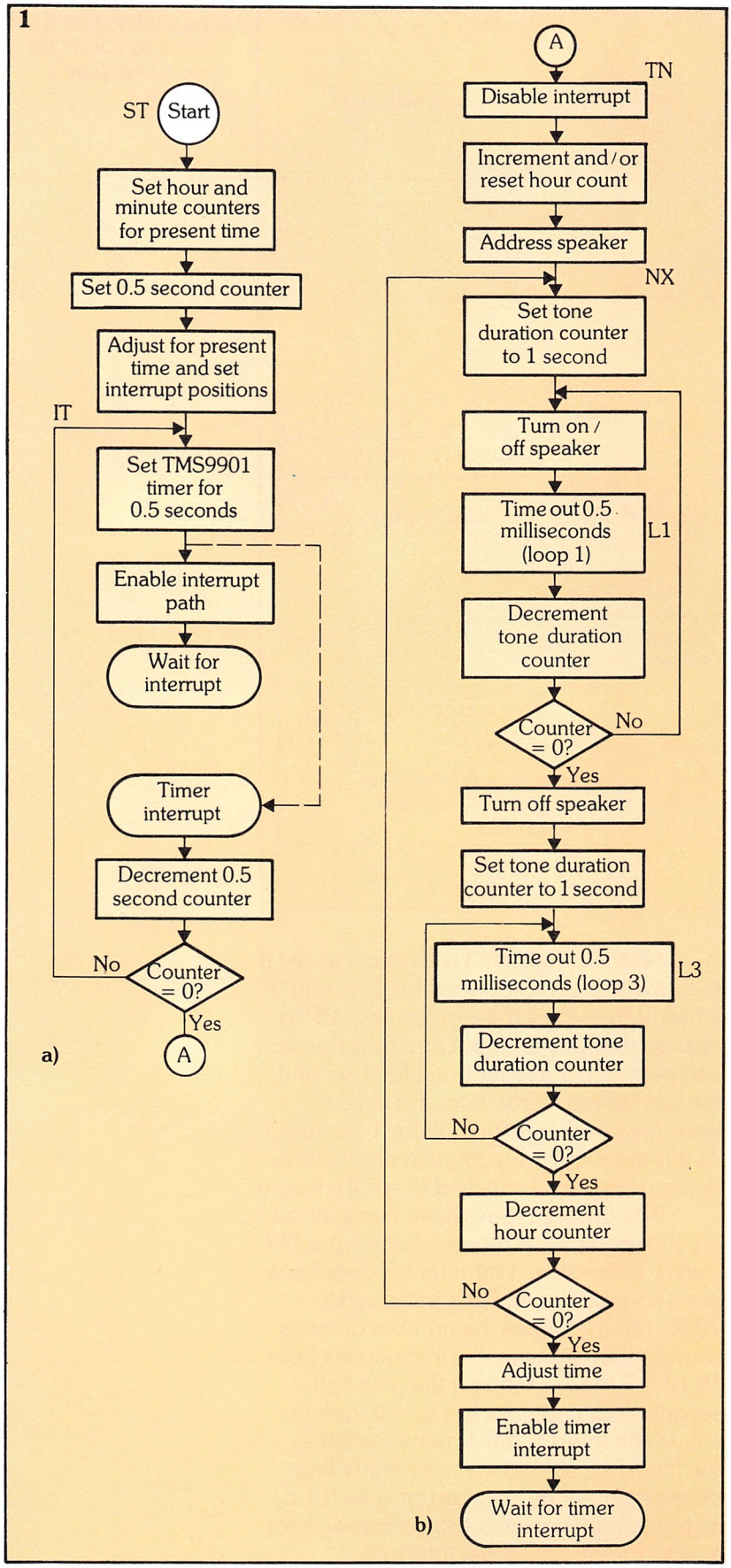
Most of the program's data requirements are associated with the counter and timer requirements. *Figure 2* shows one possible assignment for the workspace registers and counter functions.

Registers 0 to 5 contain the basic timer and interval values used for control. Register 6 holds the hour counter, while register 5 holds the hour value – you'll notice that these are identical, but are duplicated as they are used in different parts of the program. Register 7 holds the current minute count, when the time is initially set.

Register 12 must be used to hold the CRU address value, and registers 13 to 15 are used to hold the return values for the workspace pointer, program counter and status register, that existed at the time of the 0.5 second interrupt (interrupt level 1).

The workspace is located in RAM addresses 300_{16} to $31E_{16}$. The time setting program is assumed to start at location 200_{16}, while the '9901 interrupt service subprogram starts at location 226_{16} in RAM.

1. Clock system flowchart: (a) time set and clock subprogram; **(b)** tone subprogram.



The program

The program (*figure 3*) is a relatively simple implementation of the flowchart. As you can see, it begins with the time setting sequence at the label ST (set), which is at location 200_{16}.

The programmer defines the values for HOUR and MIN (minutes) before executing the program. Remember, hours and minutes values greater than 10 have to be converted to their hexadecimal equivalents. Then the programmer loads the program counter with 200_{16} and the workspace pointer with 300_{16}, and then presses the execute command button on the keyboard. We shall discuss this later.

Once the programmer has initialised the current hour and minutes values into R5 and R7, the LI 9,120 and MPY 7,9 instructions convert the minute times into 0.5 seconds. The multiplication of two 16-bit numbers like this, results in a 32-bit product, which is held in R9 and R10. R10, the least significant pair, is moved to R7, which then holds the current number of half seconds. R0 is then initialised to 7200 (the number of half seconds in an hour). The current half second value is then subtracted from R0 to get the half seconds remaining in the hour.

The next four instructions load the workspace pointer value of 300_{16} into memory location 4 and the program value of 226_{16} into memory location 6. The next interrupt level 1 procedure uses these values in jumping to the sequence, beginning at IT (interrupt).

When the timer causes a level 1 interrupt, the 300_{16} stored in location 0004 and the 226_{16} stored in location 0006 of RAM cause the microcomputer to start executing instructions at location 226_{16} (label IT), using the workspace starting at location 300_{16}. In the IT sequence the timer is re-initialised and the interrupt is decremented. If it is not zero, IDLE causes the program to wait for the next interrupt. If the 0.5 second counter is zero, then the program jumps to the tone generating sequence that starts at TN (tone).

Tone subprogram

The interrupt is disabled at TN with the LIMI 0 instruction and then the hour count is incremented. If it has been incremented

2

General use	Memory location (starting address)
Time set and clock subprogram	200_{16}
Tone subprogram	250_{16}
Workspace	300_{16}
Interrupt transfer vector	
(WP)	0004
(PC)	0006

a)

Workspace register	Address	Use
R0	300	1/2 counter 7200_{16}
R1	302	TMS9901 interval timer value $3D09_{16}$
R2	304	0.5 millisecond timer value
R3	306	1 second counter 2000_{10}
R4	308	Speaker control
R5	30A	Hour counter
R6	30C	Hour counter copy
R7	30E	Minutes-set register
.	.	
.	.	
.	.	
R12	318	CRU address
R13	31A	Main program workspace pointer value
R14	31C	Main program program counter value
R15	31E	Main program status value

b)

2. (a) TM 990/U89 RAM memory allocation; **(b)** workspace register allocation.

to 13, it is then reset to 1 with the next three instructions.

The hour count is copied to R6 and used as a counter for the number of tones, thus preserving R5. Register 12 is set to the speaker address ($43C_{16}$) and register 4 is initialised to a pattern of alternating 1s and 0s ($AAAA_{16}$). The tone duration is set at one second in R3 with LI R3,2000. At RP (repeat), the LSB of the first byte of R4 (bit 7 which is initially a zero) is sent to the speaker with the instruction LDCR R4,1. Register 4 is then circulated (SRC 4,1) to complement the LSB (now bit 7 is a 1).

The counter R2 determines the time of the tone's half period. With 22_{16} loaded in R2, added to the time it takes to execute the instructions from RP to LI2,22, the half period time is 0.5 ms. The instruction loop beginning at L1 (loop 1) is used to time this period (either on or off) to provide the 1 kHz tone.

After L1, register 3 is checked to see if the tone has been sounded for 1 second. If it hasn't then the program jumps to RP to change the speaker input and time the next half period. If it *has* sounded for 1 second, the next seven instructions are used to keep the speaker turned off for 1 second. At this time, the hour count in register 6 is decremented and checked to see if it is zero.

If insufficient tones have been sounded, the sequence is repeated starting at NX (next). If the correct number of tones *have* been sounded, then R0 is initialised to 7200, minus 4 times the number of tones sounded. In this way, the instructions from MOV 5,7 to S 7,0 correct the next half second hour count for the time it took to sound the tones. LIMI 1 then re-enables the interrupt, so that the next '9901 time interval generates an interrupt. The IDLE instruction then causes the microprocessor to wait for the next timer interrupt.

3. Program for the clock system.

3

```
ST   LI    5,HOUR     Set R5 with current hour value at 202(16)
     LI    7,MIN      Set R7 with current minutes value at 206(16)
     LI    9,120      Set R9 to ½ seconds in a minute
     MPY   7,9        Multiply current minutes by 120 for ½ seconds
     MOV   10,7       Move current ½ seconds (in R10) to R7
     LI    0,7200     Initialise R0 with number of ½ seconds in hour
     S     7,0        Subtract current ½ seconds to get those left
     LI    9,>300     Load R9 with WP value to be used by interrupt
     MOV   9,@4       Send this value to transfer vector location
     LI    9,>226     Load R9 with PC value for interrupt sequence
     MOV   9,@6       Send this value to transfer vector location
IT   LI    12,0       Set R12 to 0 (CRU base address for 9901 timer)
     LI    1,>7A13    Load R1 with timer access (bit 15 a one) and timer value of
                      3D09(16) in bits 1 through 14
     LDCR  R1,15      Send R1 contents to CRU to initialise timer to ½ second
     SBZ   0          }
     SBO   3          } Enable user 9901 timer interrupt
     LIMI  1          }
     DEC   0          Decrement ½ second counter R0
     JEQ   TN         If zero, go to tone sounding sequence
     IDLE             If not, wait for next interrupt
TN   LIMI  0          Disable timer interrupt
     INC   5          Increment current hour count
     CI    5,13       Compare hours to 13
     JLT   OK         If less than 13, hours OK
     LI    5,1        If not, reset hour count to 1
OK   MOV   5,6        Copy hour count to R6
     LI    12,>43C    Set up speaker address in R12
     LI    4,>AAAA    Set alternating 1/0 pattern in R4
NX   LI    3,2000     Set 0.5 millisecond counter (tone duration)in R3
RP   LDCR  4,1        Send bit 7 of R4 to speaker
     SRC   4,1        Shift bit pattern in R4 to complement bit 7
     LI    2,>22      Set speaker on or off time to 0.5 milliseconds
L1   DEC   2          Decrement speaker on/off counter
     JNE   L1         If not zero, continue timing
     DEC   3          Decrement tone duration counter
     JNE   RP         If not zero, continue sounding tone
     SBZ   0          If zero, turn speaker off
     LI    3,2000     Set tone off counter to one second value
L2   LI    2,>22      Set up 0.5 millisecond timer value
L3   DEC   2          }
     JNE   L3         } Implement 1 second time delay
     DEC   3          }
     JNE   L2         }
     DEC   6          Decrement hour count
     JNE   NX         If not zero, sound another tone
     MOV   5,7        If zero, move number of tones sounded to R7
     LI    9,4        Initialise R9 with number of ½ seconds per tone
     MPY   7,9        Multiply by number of tones to get time used by tones
     MOV   10,7       and get this value into R7
     LI    0,7200     Load R0 with number of ½ seconds per hour
     S     7,0        Subtract time used by tones to get ½ seconds remaining into R0
     LIMI  1          Turn on timer interrupt
     IDLE             Wait for timer interrupt
```

Running the program

The TM 990/U89 microcomputer's keyboard commands, supported by its monitor program handle the loading of the program into memory, initialising the memory locations and starting the program execution. The monitor offers several program execution and debugging capabilities that allow a program to be verified and implemented with a minimum of time and effort.

The first step in setting up the TM 990/U89 microcomputer module is to connect the power supply. The TM 990/U89 needs three power connections: +5 V, +12 V and –12 V. When the power supply is turned on, the computer should respond by sounding a beeping tone and displaying:

CPU READY

The procedure to enter the programs and constants into the microcomputer is as follows:

1) To get the computer ready to respond to a command press the Ret (for return) key and the display then shows:

?

2) The first instruction in the program is then entered by keying in A200 followed by Ret. This sets up the address 200_{16} to hold the instruction that follows. At this point the display will show:

0200

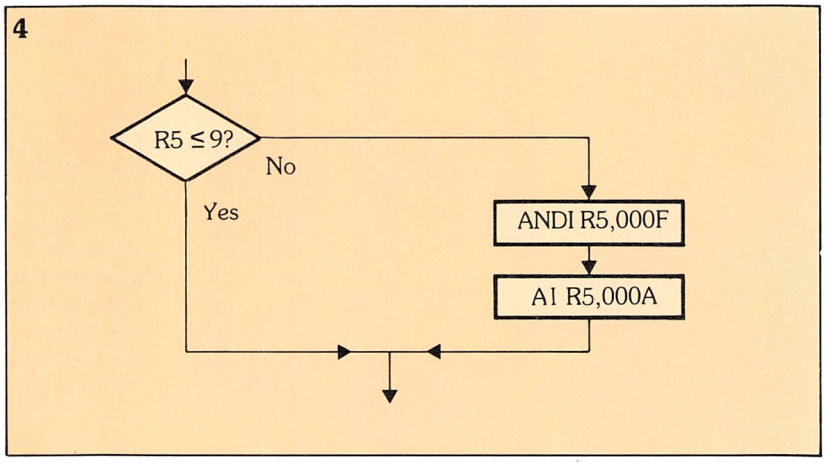
The label for that address can then be entered as a two letter symbol – in this case ST for SET. Once the label has been entered, then the mnemonic is keyed in. Here, the first instruction is L1 5, HOUR. If the instruction doesn't have a label, then it is 'omitted', by pressing the space key followed by the mnemonic instruction.

3) The assembled code corresponding to each instruction has now been keyed in and followed by Ret. Successive presses of Ret will get the address of the next instruction location. The rest of the instructions and their assembled codes can be entered in this way.

4) When the complete program has been keyed in, the current time data is entered. This is achieved by pressing the M (memory location) key followed by the address code and the data code. To place the hour value in location 202_{16} and the minute value in location 206_{16}, the following sequence is used:

M202
Ret
Hour value
Space
Space
Minute value
Ret

5) This complete, the obvious thing to do is run the program, before the time changes. The program is started by setting the program counter to 200_{16}, the workspace pointer to 300_{16}, and then pressing the E (execute) key:



4. Algorithm for the programming exercise.

P
Ret
200
Ret
W
Ret
300
Ret
E
Ret

The program then runs without any further control unless the time or program sequences have to be modified.

6) To stop a program, or to re-key a program, a system reset command is entered.

Improving and expanding performance

Our basic clock program is only useful for sounding the hour chimes. However, now the basic time counting program has been created, it would be a relatively simple matter to write a program to maintain an hours, minutes and seconds display. The

range of clock applications can also be expanded with the incorporation of timer control facilities. The microcomputer module can then be used to control and time the operation of other electrical systems.

A decimal to binary conversion subprogram could also be incorporated in the system, enabling the user to enter the current hour and minute values in familiar decimal form. A further refinement would be the incorporation of different tone generating subprograms, that would play tones, or vary the tones according to the time of the day. The programming would be similar to that which we have already seen, but more extensive timing sequences would be necessary.

Remember, because the programs that run on the TMS 990/U89 microcomputer will also run in the other '9900 family devices, the clock system could be easily implemented with a TMS9940 single chip microcomputer, and a few extra components.

To sum up . . .

1) 16-bit microprocessors and microcomputers are needed in high performance systems requiring high numerical accuracy or high speed communications.
2) 16-bit microcomputers are chosen for applications demanding high performance with minimum assembly costs.
3) 16-bit microcomputer modules are chosen for high performance systems that are to be produced in limited quantities.
4) Microcomputer modules save the user time in developing both the system hardware (it's already built) and software.

As practice, you may find it useful to run through the following exercise. Write a subprogram for a clock system that will carry out the decimal to binary time value conversion that we mentioned earlier. The subprogram is to be called CONVERT, and it will convert the hour BCD value (0 to 12) held in R5, and the BCD minutes value stored in R7 to their binary forms.

A simple algorithm for this subprogram is shown in *figure 4*. If the contents of R5 are less than 10, the number in R5 is correct; if the contents of R5 are 10 or more, then remove the tens digit and add A to the units digit.

A comparable algorithm exists for converting a two digit decimal code to its binary equivalent. The only difference is that A is added to the units digit by a number of times, given by the value of the tens digit. Again, the tens digit is removed from the number before these additions are performed. The solution to the problem is held in *figure 5*.

Try not to look at it until you have completed your attempt!

When this subprogram is completed, it may be added onto the end of the main program by inserting the instruction BL CONVERT after the first two instructions of the times set and clock subprogram.

5. Solution to the programming problem.

5

Solution

```
CONVERT   MOV   5,8      Copy hour value into R8
          ANDI  5,>F     Remove tens digit from contents of R5
          ANDI  8,>FO    Remove units digit from contents of R8
          CI    8,>10    Compare tens digit in R8 to 1
          JNE   OK       If digit 0, number in R5 is binary equivalent
          AI    5,>A     If not, add A to R5 to get binary equivalent
OK        MOV   7,8      Copy minutes value into R8
          ANDI  7,>F     Remove tens digit from contents of R7
          SRL   8,4      Get tens digit into least four bits in R8
LOOP      CI    8,0      Is tens digit 0?
          JEQ   FIN      If it is, conversion of minutes is complete
          AI    7,>A     If not, add A to contents of R7
          DEC   8        Decrement tens counter in R8
          JMP   LOOP     Go check current value of tens counter
FIN       RTWP           Conversion complete, return to calling program
```

> means base 16 constant

BASIC THEORY REFRESHER

ELECTRICAL TECHNOLOGY

Incorrectly terminated transmission lines

So far in our consideration of transmission lines, we have only looked at those that have been correctly terminated (i.e. terminated with a resistance equal to the line's characteristic resistance) and fed with a perfect voltage source. What happens if one or both of these two conditions are not satisfied?

Line terminated in a short circuit

As a first step to a solution of this problem, consider the transmission line shown in *figure 1a* of length, l, with its far end short circuited. A voltage generator gives a short rectangular pulse, of amplitude E and duration t, as shown in black in *figure 1b*. Since the pulse is connected directly to the line of characteristic resistance, R_o, the current flowing from the generator in the direction shown by the black arrow, is I_i. This incident current has a magnitude E/R_o, as shown in black in *figure 1c*.

As we know from the previous *Basic Theory Refresher*, this pulse of voltage and current travels down the line towards the termination with a velocity of propogation, u. It arrives at the termination after a time $T = l/u$ and the voltage and current are shown by the broken line pulses in *figures 1b* and *1c*. The incident voltage, v_i, and current, i_i, are related at all points from the generator to the termination through the characteristic resistance by:

$$v_i = R_o i_i$$

However, at the end of the line (since it is short circuited) the total voltage must be zero. As a consequence, another voltage, *equal and opposite to the incident voltage*, of magnitude -E must be set up there; this is termed the **reflected voltage**, v_r. This reflected voltage must have an associated current pulse:

$$i_r = \frac{v_r}{R_o}$$

Both pulses are shown by the red arrows in *figure 1a*.

Figure 1b shows, in red, the reflected voltage pulse and *figure 1c* shows, also in red, the reflected current pulse.

We may calculate the total voltage at the termination, simply by addition:

$$v_i + v_r = E + (-E) = 0$$

which stands to reason, as the termination is a short circuit.

Similarly, we may calculate the total current at the termination (note that the reflected current is in the opposite direction to the incident current):

$$i_i - i_r = \frac{E}{R_o} - \frac{(-E)}{R_o} = \frac{2E}{R_o}$$

that is, twice the incident current. The total voltage and current waveforms at the generator (in black) and at the termination (in red) are shown in *figures 1d* and *1e*.

In summary, we can say that if a voltage pulse is sent down a transmission line with a short circuit termination, we find that, after a time 2T, a reflected pulse of opposite sign will be observed at the generator associated with a current pulse of the same sign as the incident pulse.

Line terminated in an open circuit

We may carry out a very similar argument for a transmission line terminated in an open circuit. In this case, however, the total current at the termination (not the total voltage) must be zero at all times. In *figures 2a* and *2b* the total voltage and current waveforms, up to the time 2T, are shown – these exist at the generator and termination of such a transmission line.

As we would expect, the reflected voltage is now of the same sign as the incident voltage and the current pulse is reversed.

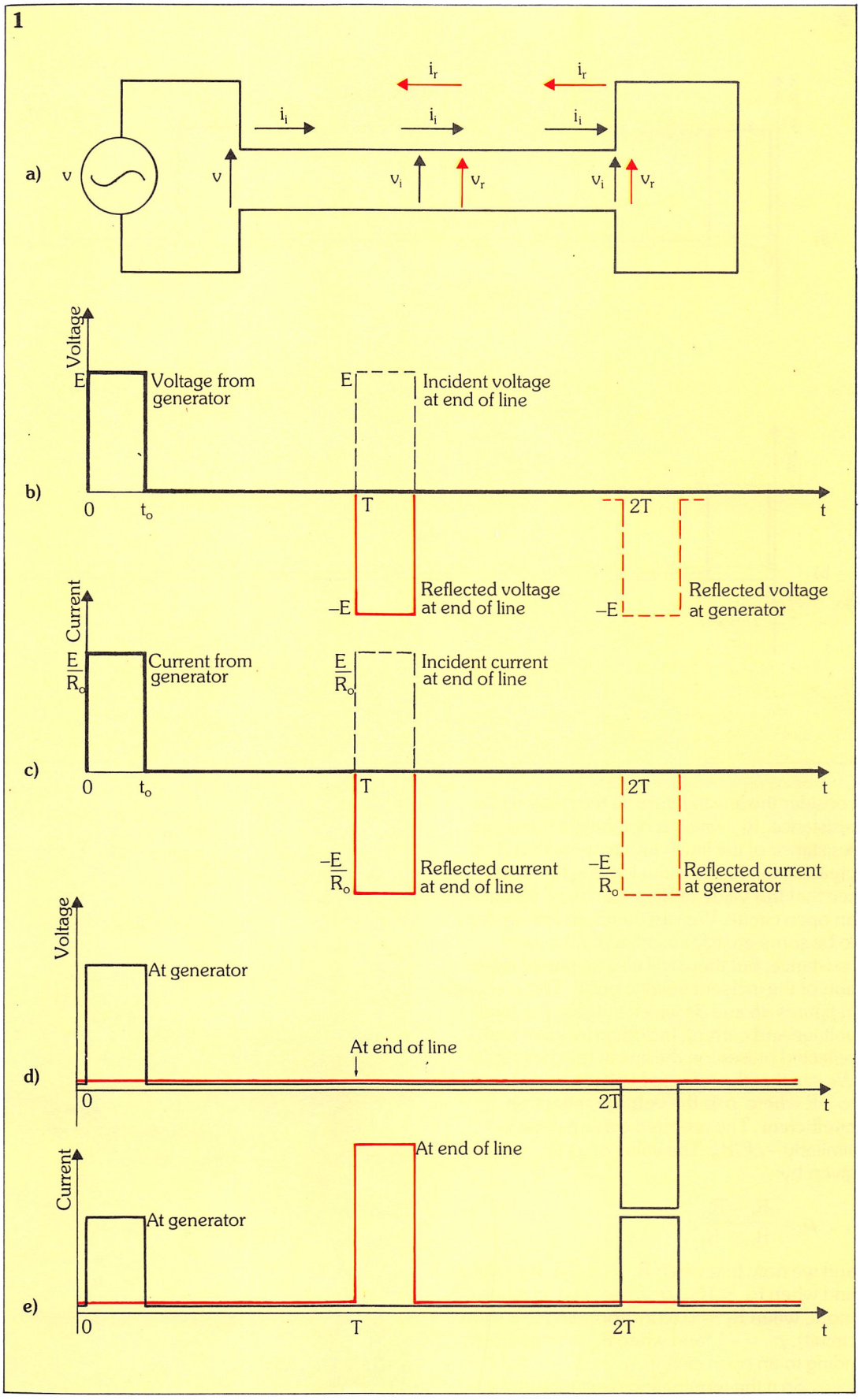
Further reflections

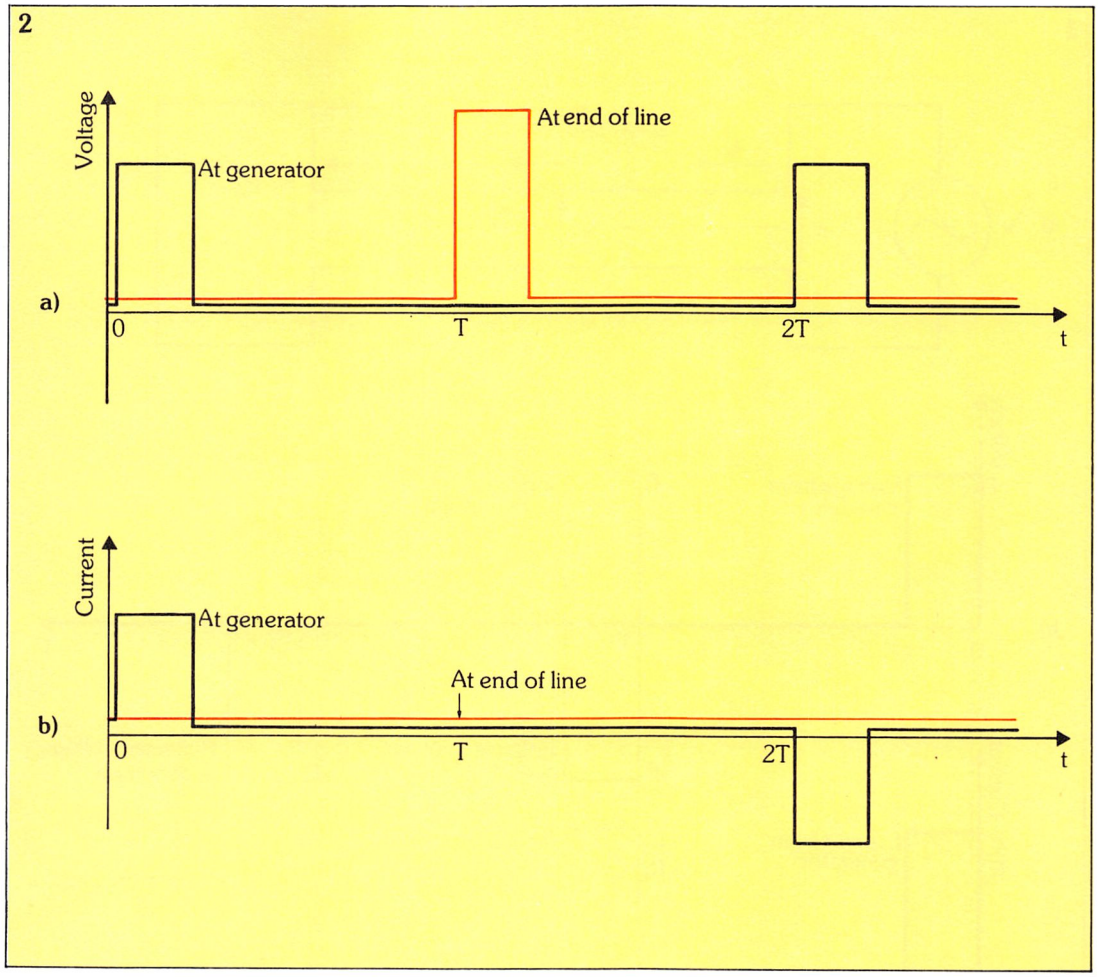
So far we have only looked at times up to 2T. What happens after this is governed by the nature of the voltage generator. We have considered the case where the supply is a perfect voltage pulse of short duration, t_o. After this time the voltage falls to zero – which corresponds to a short circuit. As a consequence, the reflected pulse arriving back at the generator sets up a second incident pulse starting at time 2T after the initial pulse. In fact, if the line has no losses, is short circuited at one end and is supplied by a short voltage pulse, there will be an infinite sequence of pulses travelling backwards and forwards along the line. In practice, of course, small losses in the line cause the pulses to die away with time.

Line terminated in a resistance

As a final example of practical significance,

1. Determining what happens when a transmission line terminated in a short circuit is fed with a perfect voltage source.





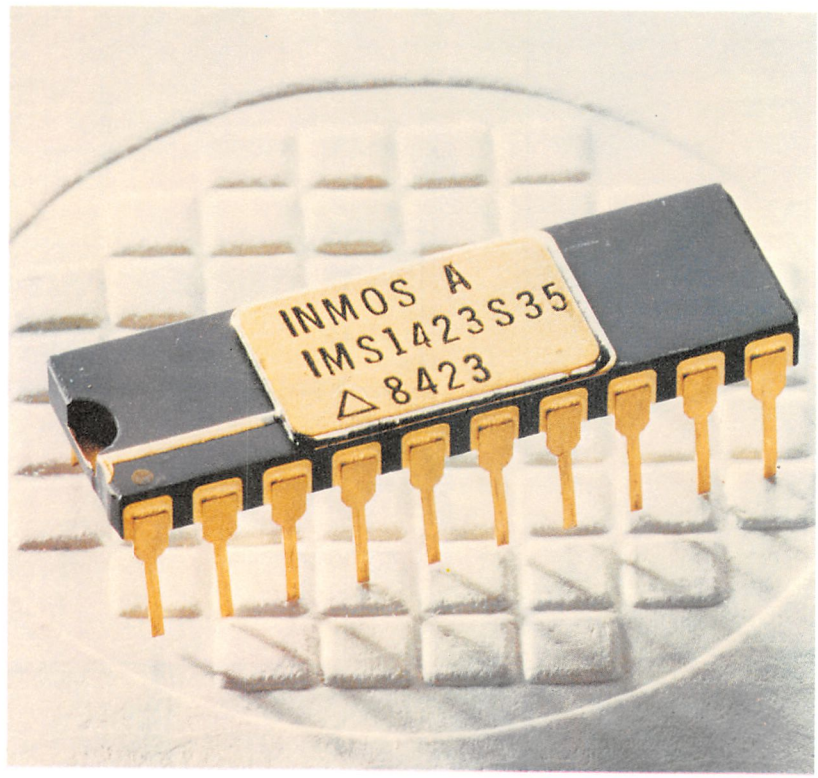
2. Determining the reflected voltage for a transmission line terminated in an open circuit.

consider the line in *figure 3a* terminated in a resistance, R_L, which is *not* the characteristic resistance of the line – let's assume that R_L is greater than R_o. Without looking too deeply we see that this value of R_L is 'part way' towards an open circuit. We can therefore expect there to be *some* energy absorbed by the load resistance; but there will *also* be partial reflection of the incident voltage pulse. This is shown in *figures 3b* and *3c* which indicate the total voltage and current, including incident and reflected pulses, on the line at the generator. We see that the reflected voltage pulse is equal to ρE where ρ is the **voltage reflection coefficient**. The reflected current pulse is similarly $-\rho E/R_o$. The value of ρ is given by:

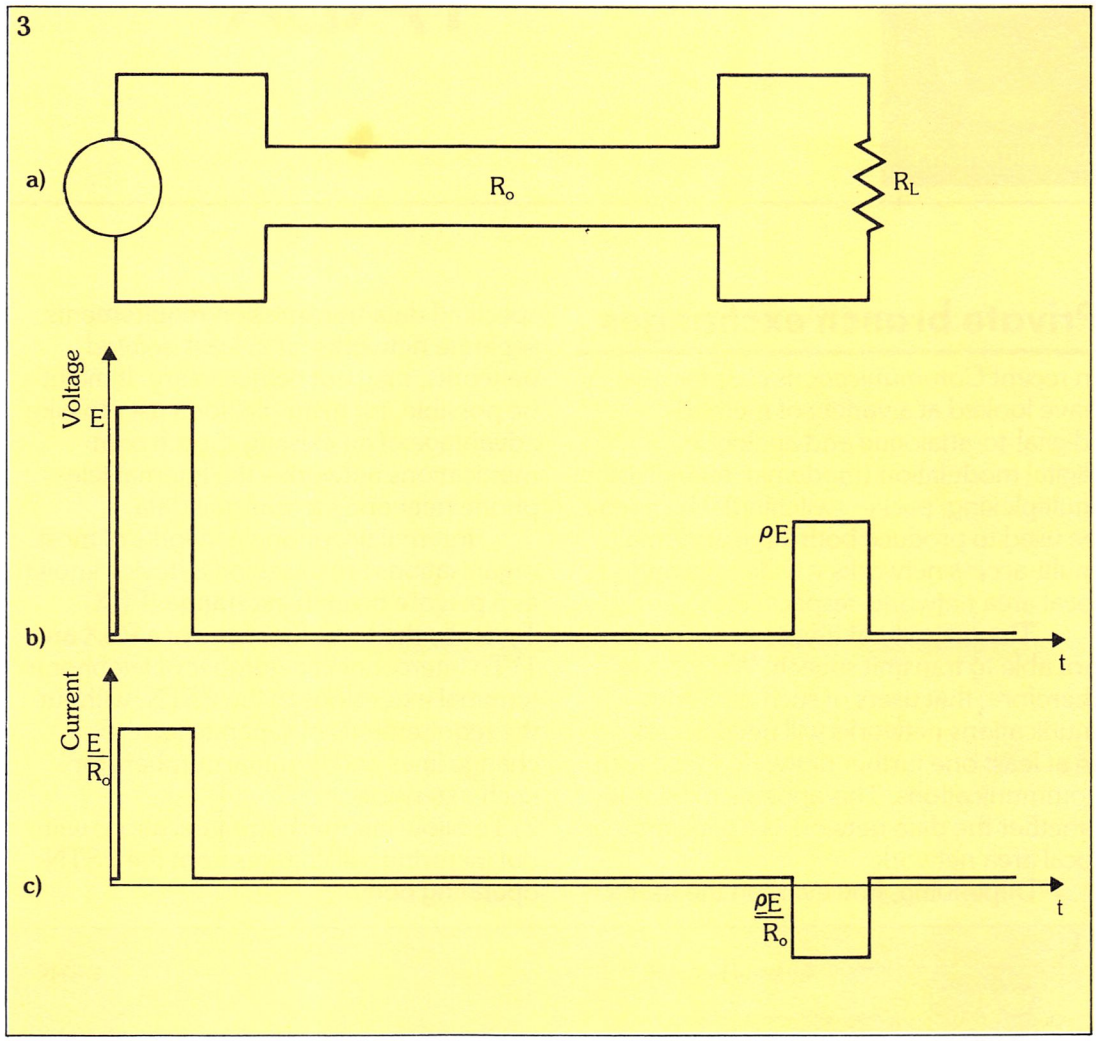
$$\rho = \frac{R_L - R_o}{R_o + R_L}$$

and we note that when $R_L > R_o$, ρ is positive; and when $R_L < R_o$, ρ is negative. Furthermore, when $R_L = 0$ (corresponding to a short circuit), $\rho = -1$, and when $R_L = \infty$ (corresponding to an open circuit), $\rho = 1$.

So if the terminating resistance is not



Below: the IMS1423 static RAM chip. INMOS' first commercial CMOS product.
(Photo: INMOS)



3. Reflected voltage for a transmission line terminated in a resistance.

exactly equal to the characteristic resistance, a small reflected wave is sent back to the generator and, of course, this will be again reflected at the generator, arriving at the termination after time 3T, and so on.

If this transmission line formed part of a computer and the initial pulse represented a logic 1, this would be received as desired, although delayed by time T. However, 2T after this there is a *second*, smaller pulse which might be sufficiently large for the computer to interpret as another, spurious pulse.

Reflected pulses are also important in the study of the effects of lightning flashes which strike overhead power transmission lines. Since power transmission lines are *never* operated with a matched load resistance, there will inevitably be multiple reflections. This may cause very high voltages to be developed on the lines. □

19

COMMUNICATIONS

PABX

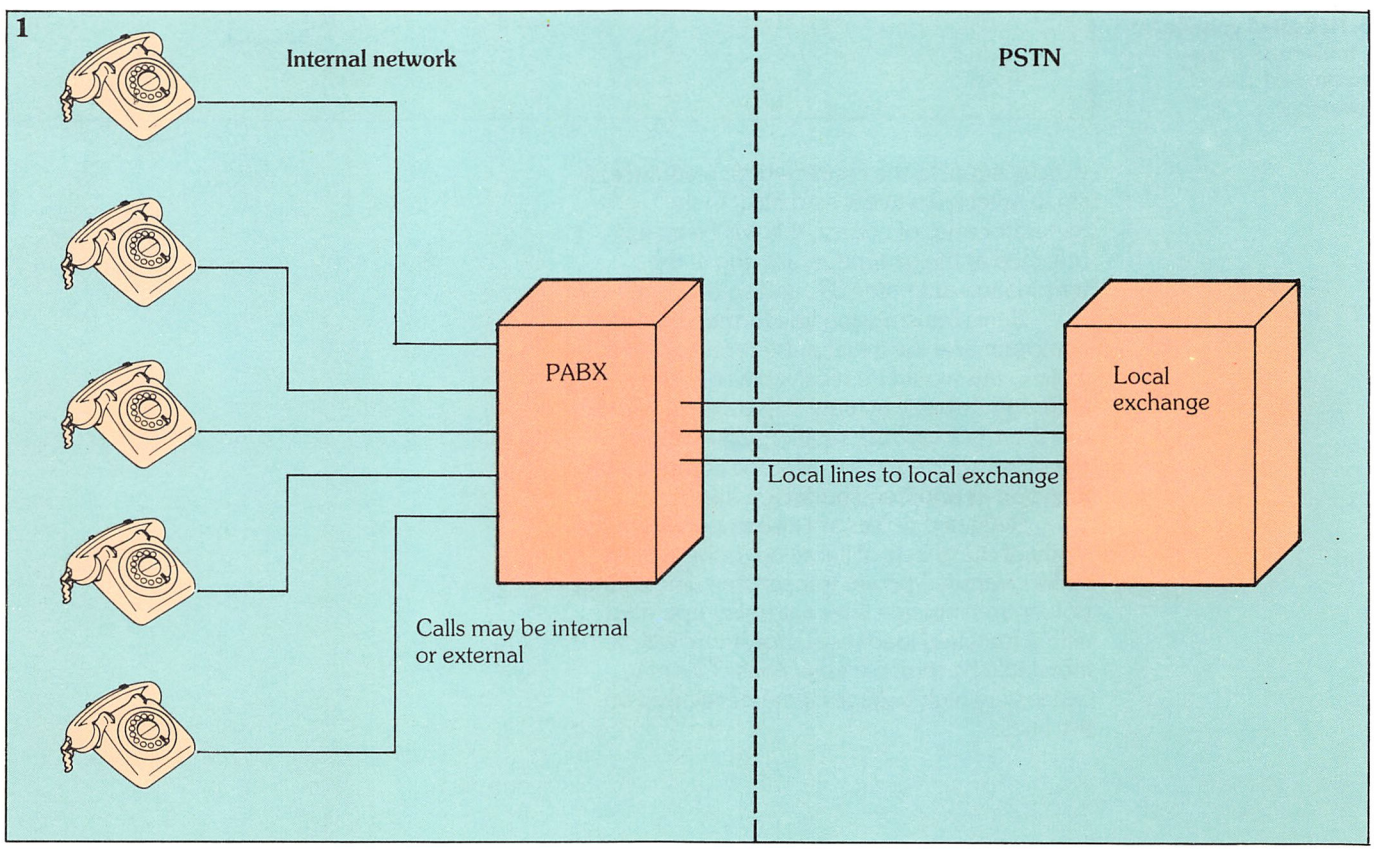
Private branch exchanges

In recent *Communications* chapters, we have looked at a variety of methods (digital-to-analogue and analogue-to-digital modulation (modems); time division multiplexing; packet switching) which may be used to produce both large and small multi-access networks – wide area and local area networks respectively.

The majority of these networks are not able to transmit speech. We can see, therefore, that users of such data communications networks will need access to at least one further network, for speech communications. This appears to be true whether the data network is a wide area or local area network.

Depending, however, on the user's specified data transmission requirements, separate networks, or at least *isolated* networks, may not be necessary. It might be possible, for example, for a user to take advantage of an existing speech communications network – the internal telephone network – to transmit data.

Internal telephone networks of most organisations are based on a device known as a **private branch exchange** (PBX). Typically the basic functions of a PBX are:
1) To interconnect a number of telephone terminal extensions to the PSTN, without the requirements of separate local exchange lines and terminal numbers for each extension.
2) To allow internal communications without incurring call charges from the PSTN operating body.

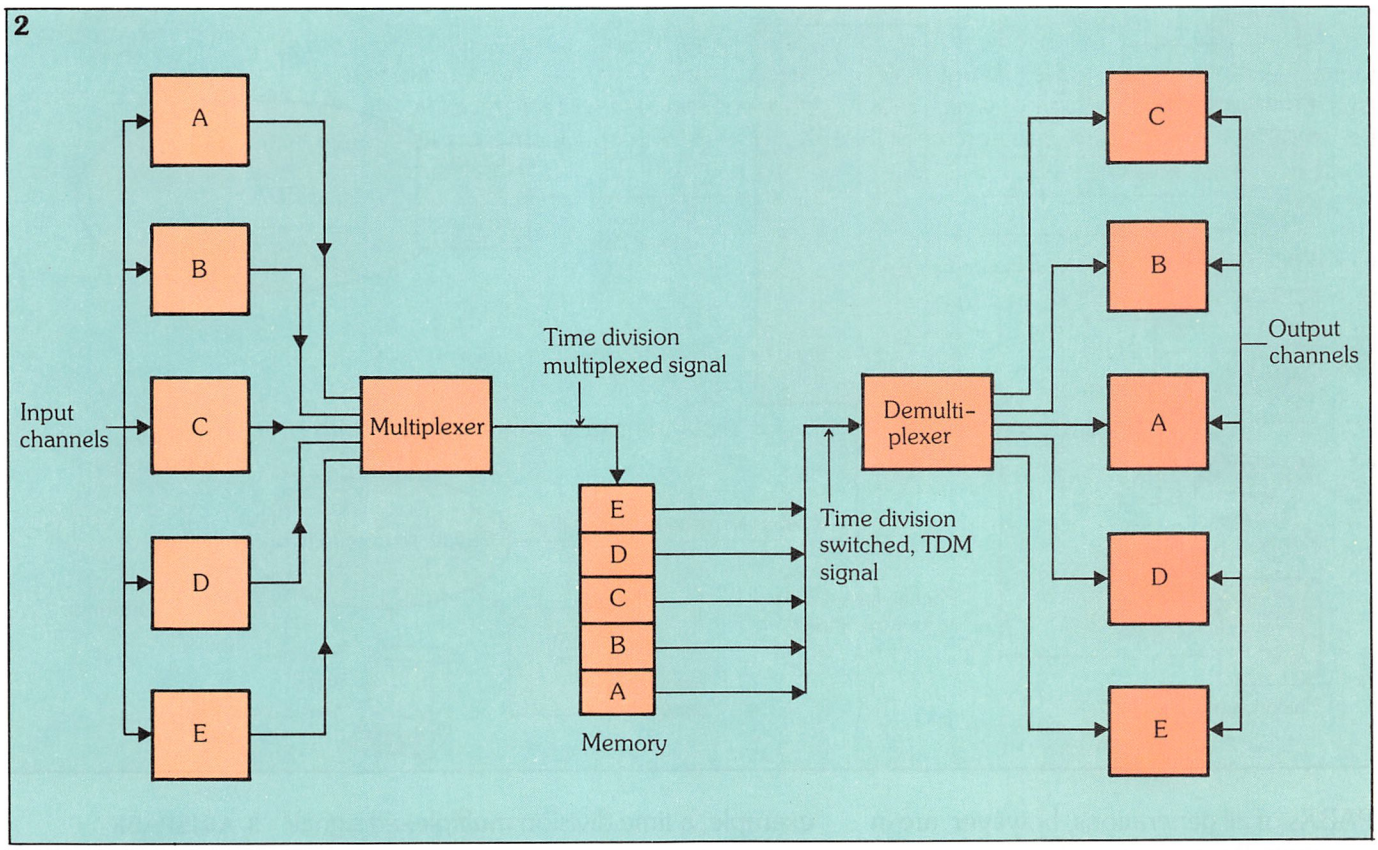


1. A PABX based internal telephone network.

Table 1
The five generations of PABX

Generation	Type	Description
1	Strowger	Analogue, space division switched
2	Crossbar	Analogue, space division switched
3	Reed relay	Analogue, space division switched, may be SPC*
4	Thyristor, solid-state	Analogue, space division switched, SPC
5	Digital, solid-state	Digital, time division switched, SPC

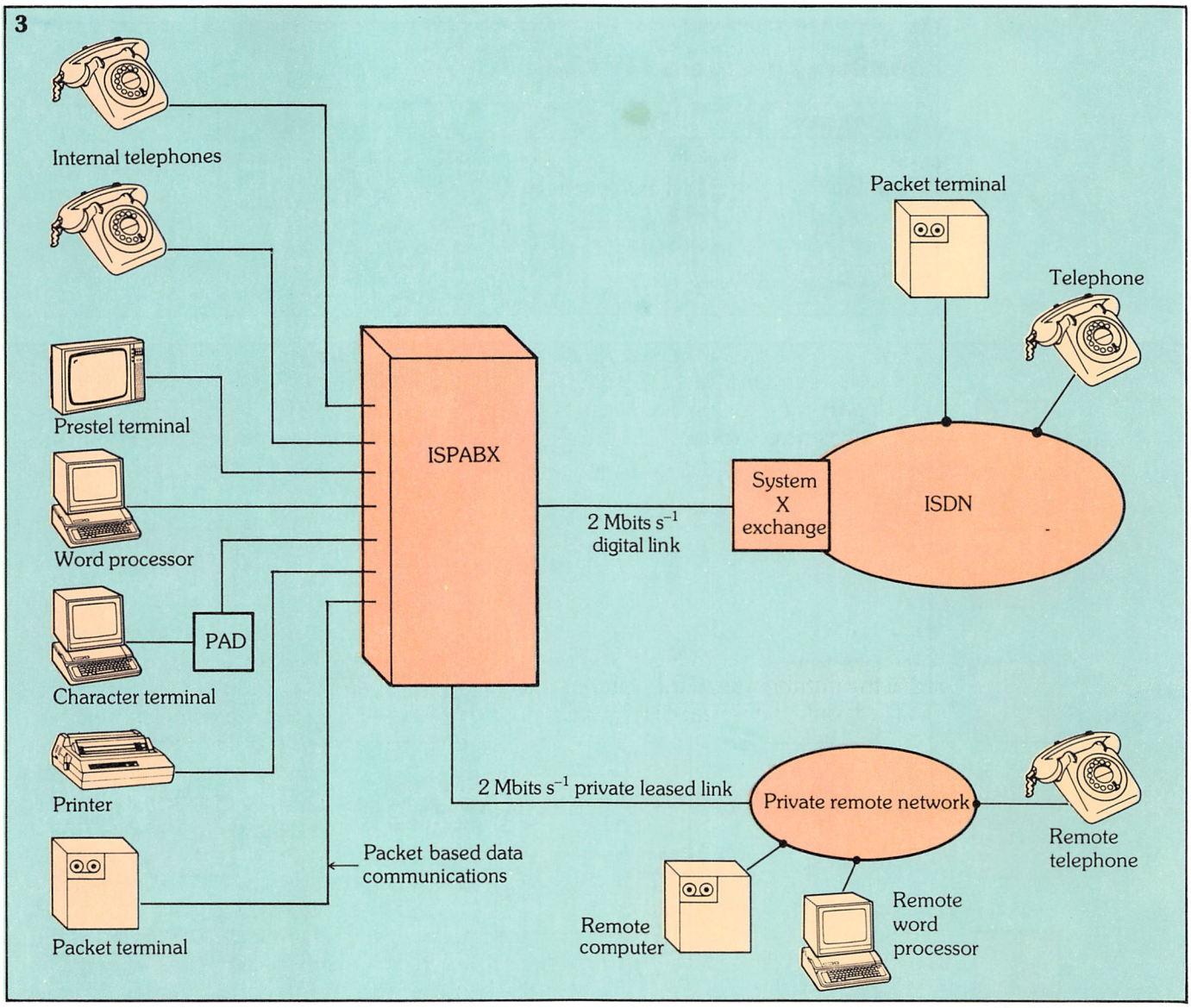
**SPC–stored program control*



2. Time division switching.

There are two main categories of PBX: the **private manual branch exchange** (PMBX) – where a switchboard operator connects all calls; and its successor, the **private automatic branch exchange** (PABX). A PABX still has a switchboard but the operator generally only has to connect incoming calls from the PSTN to the terminal extension required. An example of a PABX-based internal telephone network is shown in *figure 1*. Here, a number of local lines to the local exchange are interfaced to the internal telephone terminals via the PABX itself. Each call, whether incoming, outgoing or internal, is connected via a switching matrix within the PABX to its destination. The number of simultaneous calls which any PABX may handle is dependent on the number of switches in the matrix; in this respect, a PABX performs a similar function to any exchange in the PSTN.

Historically, there have been a number of 'generations' of PABX depending on the technology used to manufacture the switching matrix: *table 1* lists these generations with switch types. The five generations correspond, naturally, with the switching mechanisms used in PSTN exchanges, although a new switching mechanism tends to be incorporated more rapidly into PABXs than in the PSTN.



PABXs of all generations, however, are in use, even though generally only generations 4 and 5 are currently manufactured.

The first four generations of PABX all transmit speech through the switching matrix as analogue signals. Switching a call is a simple job of operating switches at various positions in the matrix and, for this reason, the technique is known as **space division switching**.

By contrast, the latest generation – the **digital PABX** – transmits speech signals as pulse code modulated digital signals, in a **time division switching** technique. Time division switching, illustrated in *figure 2*, is a derivative of the time division multiplexing transmission technique we have met previously. Where, for example, a time division multiplexed signal (consisting of a number of interleaved pulses) on a high speed circuit is written in order into memory, then read from the memory in a different order, time division *switching* has occurred. Individual pulse code modulated speech signals are switched from one channel to another, simply by controlling the order in which the memory is read.

PABX generations 3, 4 and 5 (reed relay, thyristor and digital) may all be controlled electronically by computer, in a process known as **stored program control** (SPC). This has the advantage that any changes in requirements, facilities, extension numbers etc. may be undertaken simply by altering the controlling software

3. An ISPABX – integrated services PABX – speech and data network.

and not the physical wiring or switching.

Typical features available in an SPC PABX include:

1. **Abbreviated dialling** – normal telephone number replaced by a two-digit code.
2. **Call back** – if the called extension is engaged, the PABX will ring the caller when the extension is free. Sometimes called **camp-on busy**.
3. **Call pick-up** – any extension can answer a call to another extension, so if one extension is ringing, a user at another extension can accept the call.
4. **Call barring** – certain dial codes (e.g. 0, 010) are restricted from use, so only local calls, or non-international calls may be rung.
5. **Call diversion** – unanswered incoming calls are automatically transferred to another extension after a predetermined time.
6. **Last number redial** – the last number dialled is stored so that if a user wishes to redial the number (say if the number was engaged) only the repeat code is dialled.
7. **Secretary intercept** – a call to an extension is automatically transferred to another extension for further transfer.
8. **Conference** – a number of extensions can communicate simultaneously.
9. **Direct dialling-in** (DDI) – each extension may be accessed directly from the PABX.
10. **Night service** – designated extensions connected to incoming exchange lines, at night.
11. **Call logging** – facility whereby call details may be recorded for later observation.
12. **Extension number change** – extensions may be renumbered without rewiring PABX. Useful if offices are moved.

Using a PABX as a data network

Both thyristor and digital generations of PABX may be used to transmit data as well as speech. The thyristor PABX, being analogue, requires modems to interface data terminal equipment to the network, but the digital PABX requires data terminal access DCE.

Data transmission within the internal telephone network of an analogue PABX may be reasonably fast, say, up to 19,200 bits s^{-1} between DTE. Externally, however, via the PSTN, data signalling rates are restricted to around 2400 bits s^{-1} on dialled-up lines or 4800 bits s^{-1} (possibly 9600 bits s^{-1}) on leased lines. This compares with what we have seen in recent *Communications* chapters on data transmission over the PSTN.

With the digital type of PABX, on the other hand, data transmission over the internal network is much faster – 64,000 bits s^{-1} – thanks to the digital transmission methods used for speech communications. Again, however, data transmission over the existing PSTN is restricted to modem-type communications, with the same limits to the data signalling rate as with analogue PABXs.

Even though a digital PABX offers the advantage of fairly high internal data signalling rates, its *main* advantage wil become apparent in the future as the PSTN changes to the proposed integrated services digital network (ISDN).

The ISDN concept envisages digital PABXs directly connected to local exchanges with a 2 Mbits s^{-1} digital transmission link, therefore *external* data transmission will be digital and at a data signalling rate (on a single channel) of 64,000 bits s^{-1}. All types of data communications devices may be integrated by the digital PABX: facsimile, teletex, viewdata, digital television, packet switching equipment etc. as well as telephone terminals, allowing internal and external speech and data communications.

Future digital PABXs will incorporate the data terminal access DCE required to connect DTE. Such PABXs are known as **integrated services PABXs** (ISPABX) even though none, as yet, exist. *Figure 3* illustrates how an ISPABX speech and data network may be configured for use within the ISDN. Certain city centre PSTN exchanges have recently been changed to System X, ISDN types. Most transmission links between these new exchanges and installed PABXs are still analogue at present, which means that analogue-to-digital and digital-to-analogue converters are required at the exchange. As these analogue links are replaced by 2 Mbits s^{-1} digital links, the full benefits of digital PABXs may be realised.

PABX as an integrated network

We have seen how existing and future PABXs may be configured as a dual network, transmitting speech and data, both internally and externally. In many instances, this dual networking facility of a single *physical* network can have advantages over the alternative separate speech and data networks.

The first advantage is simply one of familiarity. Most people who work in business use a PABX for both internal and external speech communications. It functions by a well tried and tested communications technique – circuit switching. Transmitting data over the PABX is a simple idea to understand – very little user resistance to change would be expected.

In many cases, *all* the data and speech requirements of an organisation could be fulfilled by the single network of a PABX. The cost of purchasing and maintaining two (expensive) separate communications networks is obviously much higher than that of a single network.

On the other hand, the present internal and external data signalling rates of PABXs are not high. Future techniques such as the simultaneous use of two or more data channels and completion of ISDN and high speed PABX-to-exchange transmission links, however, will allow transmission of data at higher signalling rates between DTEs. Furthermore, a future generation of PABX may allow the facility of interconnection with an ultra-high speed local area network, so that the advantages of both can be utilised if required.

Glossary

direct dialling-in (DDI)	PABX facility in which individual extensions of the internal telephone network may be accessed by an incoming call without switchboard operator intervention
integrated services PABX	future type of PABX which will interface externally with an ISDN exchange via a 2 Mbits s^{-1} digital link and, internally, with both telephone terminals and all types of DTE
private automatic branch exchange (PABX)	a private branch exchange from which outgoing calls may be made automatically, without the intervention of the switchboard operator
private branch exchange (PBX)	a private exchange interfacing an organisation's internal telephone network to the PSTN
private manual branch exchange (PMBX)	a private branch exchange with which all calls, incoming or outgoing, internal or external, must be connected by a switchboard operator
space division switching	switching technique of all but digital PBXs, in which a matrix of switches allows call connection
stored program control (SPC)	name for a computer controlled exchange, in which all facilities are controlled by the program which is run in the computer. Changing the program changes the facilities
time division switching	switching technique in which interleaved parts of time division multiplexed signals are written into a store, then read in a different order. Individual pulse code modulated signals are switched from one channel to another by controlling the order in which the stored parts are read